# Virtuosity: Performing Virtual Network Resource Management

*Andrew T. Campbell*[1], *John Vicente*[2], *Daniel A. Villela*[1]
[1] Center for Telecommunications Research, Columbia University
[2] Intel Corporation
genesis@comet.columbia.edu

## Abstract

*The creation, deployment and management of network architecture is manual, time consuming and costly. To the network architect the creation process is ad-hoc in nature, based on hand crafting small-scale network prototypes that evolve toward wide scale deployment. We envision a different paradigm where 'spawning networks' are capable of profiling, spawning, architecting and managing distinct virtual network architecture on-the-fly. This paper provides an overview of a virtual network kernel and its life cycle of spawning virtual networks, and focuses particularly on the role of resource management of virtual networks. We describe, virtuosity, a virtual network resource management system that minimizes the complexity of handling multiple spawned virtual networks that operate over multiple timescales. Virtuosity is driven by per-virtual network policy exerting control and management over multiple spawned virtual networks (characterized by a set of resources) and their spawned architecture (defined as a set of interacting controllers objects) by dynamically influencing the behavior of resource controllers over slow timescales. Virtuosity provides a foundation for the management of virtual networks and forms an integral part of the virtual network kernel being developed within the Genesis Project at Columbia University.*

## 1 Introduction

The rapidly evolving nature of the application base, service demands and underlying network technology presents a significant challenge to the deployment of new network architectures. This challenge calls for new approaches to the way we design, develop, deploy and analyze next-generation network architecture in response to future needs and requirements. Currently, the creation and deployment of network architecture is manual, time consuming and a costly process. To the network architect the creation process is typically ad-hoc in nature, based on hand crafting small-scale prototypes that evolve toward wide scale deployment. We envision [6] a future communication middleware platform capable of profiling, spawning, architecting and managing distinct virtual network architecture on-the-fly.

Virtual networks can be viewed as customized communication environments [20] allowing controlled access to groups of users with specified requirements for connectivity, privacy, isolation and service demand provisioning. Recently, the growth of the Internet has given rise to the demand for more sophisticated provisioning of virtual private networks given that the existing solutions [8] tend to have limited capability and are inflexible at supporting the introduction of distinct network architecture above and beyond 'root' network architecture (e.g., the best effort IP architecture, MPLS, etc.).

We believe that the design, creation and deployment process for realizing new architectures must be automated and built on the foundations for programmable networks. The recent emergence of open programmable networks [17] [7] [1] [15] [11] is enabling new approaches to the problem of service creation and support for multiple control architectures [14][19]. This is resulting in better network customization, resource control and service delivery speed along with the flexible treatment of network traffic. In [12] we describe the process of automating the creation, deployment and management of new network architectures as "spawning". The term 'spawning' finds a parallel with an operating system spawning a child process. By spawning a process, the operating system creates a copy of the calling process. The calling process is known as the *parent* process and the new process the *child* process. Notably the child process inherits its parent's attributes typically executing on the same hardware (i.e., the CPU). We envision communication middleware associated with 'spawning networks' has having the capability to spawn not processes but complex network architectures. As the term spawning implies, we believe that the power of distributed systems technology needs to be brought to bare to dynamically create, deploy, manage and architect new network architectures. In [6] we described a *virtual network kernel* distributed across end-system and network nodes providing middleware support for spawning of distinct virtual network architectures.

A key component of the virtual network kernel is management of spawned virtual networks. In this paper, we describe *virtuosity*, a virtual network resource management system that minimizes the complexity of handling multiple spawned virtual networks that operate over multiple timescales on the same physical network hardware. Virtuosity is driven by per-virtual network policy exerting control and management over multiple spawned virtual

[1] Daniel Villela is a CNPq-Brazil Scholar
[2] John Vicente is a Visiting Researcher at Columbia University

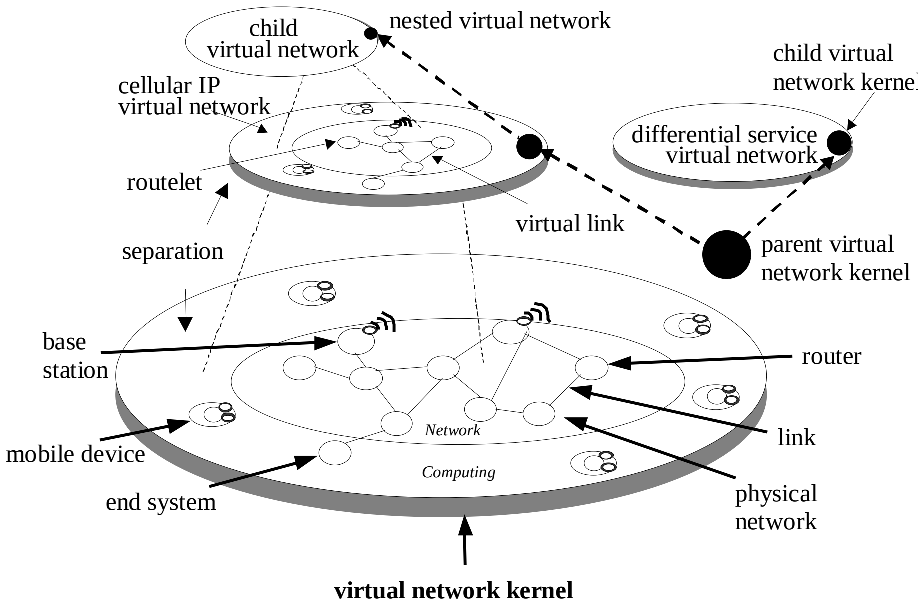


*Figure 1: Spawning Networks*

networks (characterized by a set of resources) and their spawned architecture (defined as a set of interacting controllers objects) by dynamically influencing the behavior of resource controllers over slow timescales. The structure of the paper is as follows. In Section 2, we present some background information on the virtual network kernel and the supporting virtual network life cycle. Following this in Section 3, we introduce the virtuosity framework that minimizes the complexity of handling multiple spawned virtual networks. The framework forms an integral part of the virtual network kernel capability. Design and implementation considerations are then discussed in Section 4, and we present some concluding remarks in Section 5.

# 2 Spawning Networks

## 2.1 Virtual Network Kernel

We call the virtual network kernel installed on top of the network resources the "parent virtual network kernel". We propose the realization of a parent virtual network kernel with the capability of creating a "child virtual network kernel" operating on a subset of network resources as illustrated in Figure 1. This is a departure from the operating system analogy where the parent and child typically share the same hardware.

### 2.1.1 Spawning Virtual Networks

The partitioning of the network resources and the architecture controlling those resources, i.e., the child virtual network kernel, are spawned by the parent virtual network kernel. The two architectures (i.e., parent and child) would be deployed in response to possibly different user needs and requirements. For example, part of an access network to a wired network might be redeployed as a cellular IP [21] virtual network that supports differentiated services [2][3] as illustrated in Figure 1. In this case the wired network is the "parent" and the wireless network the "child". Typically, spawned network architectures support alternative transport, signaling, traffic treatment and network management capabilities in comparison to their parent architecture. Alternatively, they may inherit the same set of distributed algorithms.

### 2.1.2 Routelet and Virtual Links

The virtual network kernel can dynamically build virtual network architectures over the physical network through the deployment of virtual network infrastructure. The parent virtual network kernel "bootstraps" the child virtual network, then creates a set of *routelets* [6] and *virtual links* that constitutes the virtual network topology. Virtual links are partitioned bandwidth segments interconnecting routelets. Routelets represent a virtual router in the virtual network topology capable of forwarding packets based on the instantiated virtual control objects. This includes a set of distributed objects that encapsulate transport (e.g., TCP, RTP) network (e.g., IPv4, IPv6) services and protocols, signaling (e.g., RSVP, in-band approaches), control (e.g., differential services, best effort) and management (e.g., SNMP, CMIP) objects. These objects and the spawning capability are encapsulated in the routelet object abstraction.

### 2.1.3 Nested Virtual Networks

The child, like its parent, inherits the capability to spawn other virtual networks creating the notion of spawning *nested virtual networks* within a virtual network. This is consistent with the idea of dynamically creating a virtual network infrastructure that supports relatively long lived (e.g., a corporate virtual network that operates over long timescales) and short lived (e.g., collaborative group networking operating within the context of the corporate virtual network but is only active for short period) virtual networks as illustrated in Figure 1. Virtual networks represent child networks (i.e., subscribers) where every child virtual network can be a parent (i.e., provider) to its own children and the depth of the *virtual network inheritance tree* (as illustrated in Figure 3) is determined by the overall availability of resources and the child network requirements.

### 2.2 Virtual Network Life Cycle

The virtual network kernel is driven by a life cycle process, as illustrated in Figure 2, that supports the dynamic creation and deployment or "spawning" of virtual network architectures through:

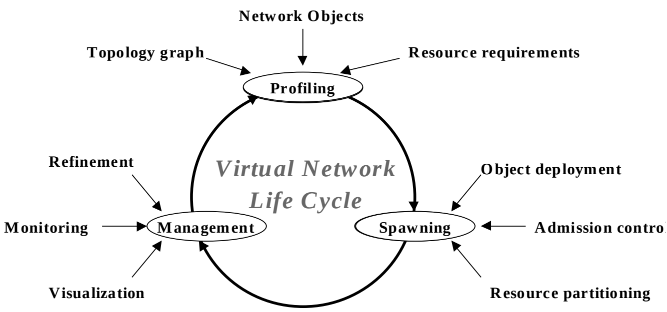


*Figure 2: Virtual Network Life Cycle*

- *profiling*, which captures the 'blueprint' of the virtual network architecture in terms of a comprehensive profiling script. Profiling captures addressing, routing, signaling, security, control and management requirements in an executable profiling script that is used to automate the deployment of programmable virtual networks;
- *spawning*, which systematically sets up the topology and address space, allocates resources and binds transport, routing and network management objects to the physical network infrastructure. Based on the profiling script and available network resources, network objects are created and dispatched to network nodes thereby dynamically creating a new virtual network architecture; and
- *management*, which supports virtual network resource management based on per-virtual network policy to exert control over multiple spawned network architectures. In addition, virtual network 'architecting' is supported, which allows the network designer to analyze the pros and cons of a virtual network design space.

# 3 Virtuosity

The virtuosity architectural model is embedded in the virtual network kernel and supports the concept of virtual network resource management through the virtual network life cycle. For full details on the virtual network kernel see [6]. Virtuosity is distinguished by its characteristics, which include *autonomous control, dynamic provisioning, capacity classes and resource management inheritance*. In what follows we discuss these characteristics and introduce the virtuosity architectural model, a novel approach to virtual network resource management.

### 3.1 Design Characteristics

Virtuosity is based on the following design characteristics:

- *Autonomous Control:* Spawning results in the composition of a child virtual network architecture, partitioning of parent network resources in support of a child's resource needs, the separation of responsibilities and transparency of operation between parent and child architectures. Once a child network has been spawned, the child has complete freedom to manage its resources and users' QoS in an autonomous manner based on its instantiated architecture. Spawning separation supports the notion of autonomous control between the parent and child architectures.

- *Dynamic Provisioning:* Virtuosity supports the dynamic provisioning of virtual network resources. Currently, the provisioning of virtual network resources is limited to either static or policy-based provisioning [18]. Virtuosity envisions a different form of provisioning where the capacity needs of individual virtual networks may change more dynamically in term of timescales and events. Virtuosity employs a per-virtual network policy-based approach that can be programmed by the

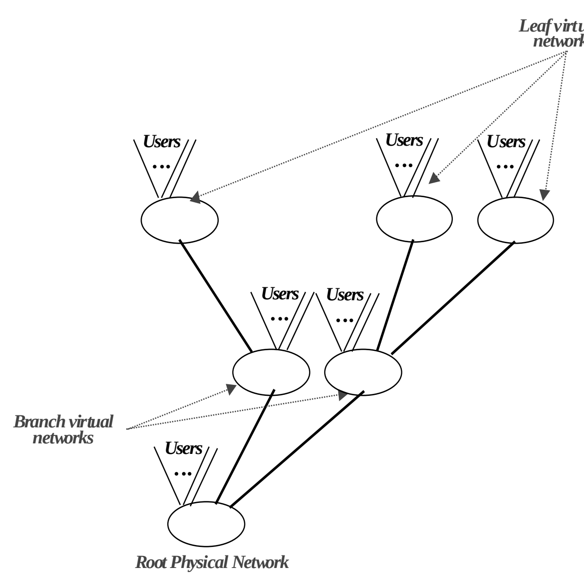


*Figure 3: Virtual Network Inheritance Tree*

subscriber to support a wide range of dynamic resource provisioning strategies.

- *Capacity Classes:* Virtual links support capacity classes within which child traffic classes (e.g., assured service, constant bit rate, best effort) are mapped and multiplexed. Capacity classes provide general purpose 'resource pipes' allowing the underlying parent controller architecture to deal with child traffic in an aggregated manner. This approach is similar to the concepts of resource management in differentiated service networks [2] but with application to virtual networks. The mapping of the child QoS to parent capacity classes is made transparent to the parent and is the responsibility of the child virtual network architecture. Virtuosity supports the following capacity classes:

  1. *constant capacity*, which statically allocates bandwidth to virtual link based on a peak rate specification providing resource isolation and an assured rate virtual link service;
  2. *controlled capacity, which allocates bandwidth based* on an average rate specification to virtual links providing resource sharing with other controlled capacity virtual links and a statistical rate virtual link service; and
  3. *best-effort capacity*, which allocates bandwidth based on some fixed amount to a virtual link for the transmission of all best effort traffic across a virtual link with no explicit service assurance.

- *Inheritance*: Through distributed object technology, resource management inheritance allows a child virtual network to transform itself to serve as a provider; giving it resource management capabilities and provisioning characteristics of its parent or, alternatively, to spawn completely distinct capabilities. Through inheritance, aggregation and the provisioning of a common set of capacity classes, virtuosity can efficiently support the resource management needs of multiple child virtual networks in the same manner that it handles new client (i.e., local end-system) resource needs. The nesting process allows us to push the complexity of the management of virtual network resources up the inheritance tree into the child virtual network, with the benefit of only having to manage reduced state information.

### 3.2 Virtuosity Framework

The elements of the virtuosity architectural model (illustrated in Figure 4), which comprise of *maestros*, *delegates*, *auctioneers*, *arbitrators*, and *monitors* are instantiated as part of the child virtual network kernel during the spawning phase and are deployed as distributed objects within routelets. As shown in Figure 4, with the exception of the arbitrator, all other elements operate in the management plane. The arbitrator operates in the data plane. Through the process of virtualization, virtual networks are separated from the physical or parent virtual network within a partitioned and separate name and address space.

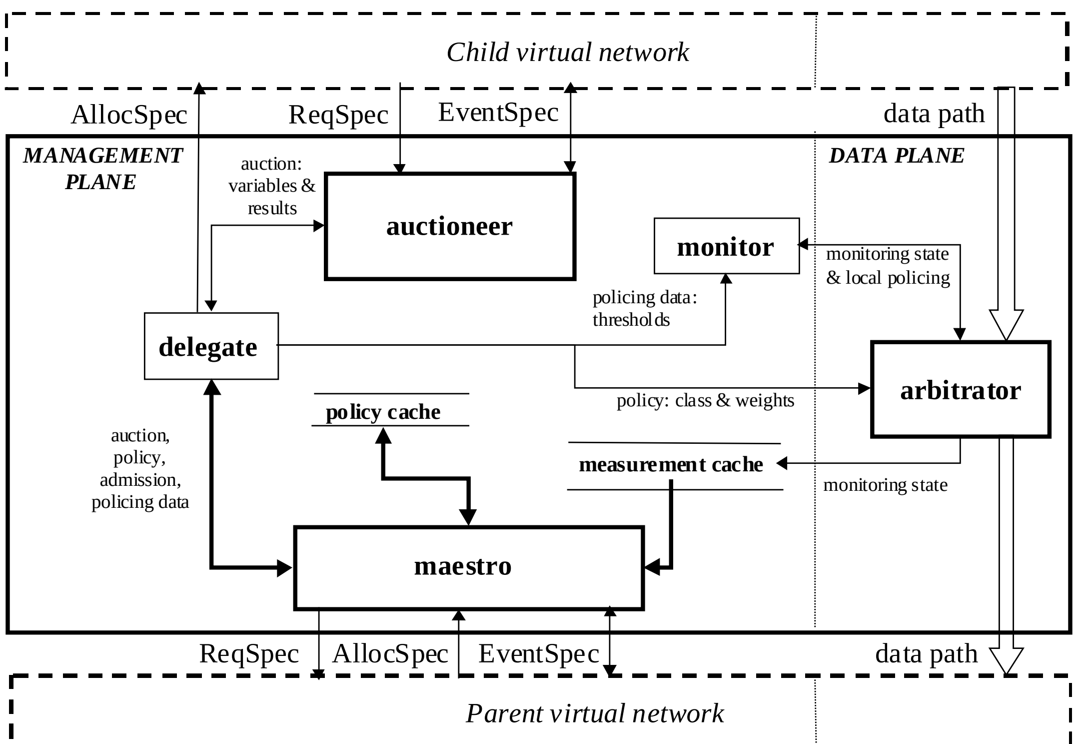


*Figure 4: Virtuosity Architecture Model*

### 3.2.1 Slow Timescale Control

Virtuosity manages and controls virtual network resources on a slow performance management [10] timescale that operates on the minutes / tens of minutes period. We argue that this is a suitable timescale for virtuosity to operate over while allowing virtual networks to perform dynamic provisioning, as needed. We believe that it is unlikely that virtual networks would operate in a stable manner for faster timescales [6]. If our assumption is correct, then virtuosity will remain fairly idle and not perturb virtual network resource management. We believe that the combination of slow timescale, dynamic provisioning, inheritance, aggregation and the adoption of small set of common capacity classes within a virtual network inheritance tree makes virtuosity highly flexible, scalable and novel.

### 3.2.2 Virtual Network Inheritance Tree

The virtual network kernel creates a natural hierarchy through partitioning and isolation of virtual networks; promoting inheritance and the autonomous control of network resources. Virtual networks are formed hierarchically through nested parent-child formations along a virtual network hierarchy tree structure as illustrated previously in Figure 3. Virtual network kernels build well organized hierarchy over the physical networks thereby reducing the complexity of spawning virtual networks and handling nested networks through inheritance and state aggregation. In addition, the virtual network kernel reduces the complexity of managing multiple virtual networks over the same physical network, which is important as the number of virtual networks increases. In this respect, we feel that the virtual network inheritance tree model is preferable over constructing virtual networks based on ‘flat’ models.

### 3.2.3 Architectural Model

The elements of the virtuosity architectural model are instantiated as part of the child virtual network kernel during the spawning phase and are deployed as a set of distributed plug-in objects. As shown in Figure 4, with the exception of the arbitrator, which operates in the data plane, all the other virtuosity elements operate in the management plane. These architectural elements are as follows:

- *maestro*, which is the key resource controller responsible for managing the global resource policy within the virtual network or virtual network domain[3]. It operates on the virtual network management timescale and is driven by current resource availability and per-virtual network policy. Maestros set pricing and rate strategies across its managed resources influencing child virtual networks in a manner to promote the efficient use of resources. A *delegate*, serving as a decentralized proxy agent for a maestro, manages all local resource interactions and control mechanisms on a routelet as illustrated in Figure 3;
- *auctioneer*, which implements an economic auctioning model for resource allocation supporting various strategies between virtual network providers and subscribers. The auctioneer services bids from child virtual networks over slow provisioning timescales promoting a competitive system among subscribers. A *monitor* performs policing and monitoring on individual parent resources. Policing assures that child virtual networks are not consuming parent virtual networks' resources above and beyond an agreed allocation of the virtual link capacity being managed;
- *arbitrator*, which represents a transport module capable of abstracting the virtual network capacity 'scheduler' controlling access to each parent resource. The arbitrator receives a virtual network scheduling policy from the maestro over slow timescale provisioning intervals upon the completion of a resource allocation process. The virtual link arbitrator manages the access and control to the parent's virtual link based on virtual network policy.

Virtuosity operates within the child and parent virtual network kernels managing the partitioned resource space and interfacing with the parent virtuosity system to increase or decrease the current partitioned resource space through dynamic provisioning. The arbitrator and monitor elements are instantiated on routelets on each port, managing the integration of provisioned capacity and local resource policy over each routelet virtual link. A single delegate and auctioneer are instantiated per routelet and manage local resource management activities on the routelet. The delegate is a coordination proxy working on behalf of the maestro to distribute local (per routelet or per port) activities, while the auctioneer brokers the provisioning (outgoing capacity classes on virtual links) requirements from multiple child networks on the routelet. Maestro is the only virtuosity element that oversees the entire resource domain. While managing (i.e., resource monitoring state and child network policy) all domain resources, it seeks optimal global policy (i.e., resource pricing, global and local resource provisioning) and distributes (via per routelet delegates) per routelet auctioning parameters and per virtual link arbitration policy. Maestro, although conceptually a centralized controller, can be *implemented* on a centralized server or decentralized using cooperating agents instantiated on a per routelet basis.

## 3.3 Maestro

The maestro is the central controller in virtuosity; it is responsible for managing the global resource policy within a virtual network or a virtual network domain and managing virtual network resource allocation. Maestro maintains global state (in a fully distributed manner) of its virtual network. It coordinates virtual network control through distributed virtuosity components performing virtual network monitoring, economic-based resource allocation

3 A domain is defined here to be a set of virtual network resources (i.e., virtual links, routelets and capacity) that the virtual network provider has complete authority for administration, pricing, control and management.

and capacity-based scheduling, all of which operate or exert control on management-level timescales.

#### 3.3.1 Dynamic Provisioning

Maestro uses dynamic provisioning of virtual network resources to meet the changing needs of its child networks (captured in child per-virtual network policy) and to react to changes in its global state; that is, it may need to respond to dynamic changes in its own virtual network (e.g., changing the resource needs of its clients), its child networks changing resource needs and its underlying parent network. The maestro interacts with its local auctioneer via a delegate to handle the dynamic provisioning of its own virtual network in relation to its child networks' needs. Upon aggregation of its clients' and child networks' demands, it interacts with its parent network auctioneer in the form of bidding for resources to support their demand requirements.

The maestro can influence the way in which resources are allocated to its child networks by setting optimal market pricing strategies [16] and resource allocation strategies, e.g., under provisioning its own virtual link resources but overbooking resources to child networks to maximize revenue for the controlled capacity traffic. Because the maestro maintains global state of its virtual network resources (e.g., being cognizant of the overload or under-use of its own virtual network links), it can influence child virtual networks to respond to this global state via optimization strategies. The maestro can, for example, influence a child network's routing mechanisms to reroute traffic along a more cost effective path (e.g., under-utilized) in the child virtual network, hence, improving the utilization

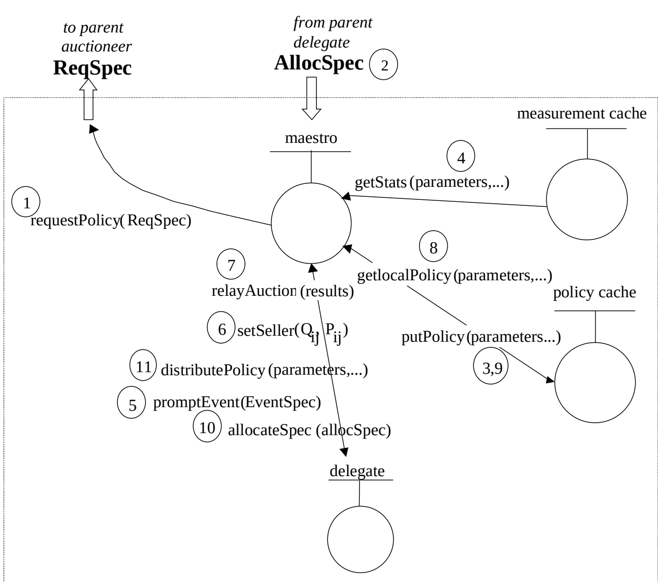


*Figure 5: Maestro Object Model*

of its own resources and potentially maximizing the revenue it can earn as a provider.

#### 3.3.2 Maestro Design

##### 3.3.2.1 Measurement-based Admission Control

During the spawning phase of a child network the maestro conducts a virtual network admission control test based on the resources requested by the child network topology. If the test is positive, then the parent provider network admits the child network and allows it to become a participant in the auctioning process controlled by the auctioneer and governed by the child virtual network's policy. Admission is coordinated by the parent maestro using its virtual network hierarchy tree. The parent maestro receives a *ReqSpec()* CORBA Interface Definition Language (IDL) (See Figure 8 in Appendix) for admission from a child virtual network and determines if sufficient resources are available within the context of its own available network resources to meet those new demands. If this is the case it indicates that the parent has sufficient residual capacity in its own right to accommodate the child's needs. Admission is based on evaluating the *ReqSpec()* target capacity class (viz. rate_quantity) against existing capacity classes. Virtuosity implements a measurement-based virtual network admission control test. By monitoring the available capacity along all of its virtual links, the maestro determines if resources allocated along its virtual links are also under utilized. Based on this measurement state information (which is maintained in the measurement cache) and capacity threshold violations (which indicate the number of times a monitored capacity has been violated) it can allocate under utilized resources based on the capacity class, bandwidth and policy requested by the new virtual network. If capacity is available, the child network is immediately admitted and the child network is allowed to participate in the auction process. In the case that the parent has insufficient resources to accommodate the new child network then it needs to renegotiate its needs with its own parent (and hence its provider) at the next level down its virtual network hierarchy tree. In this case, a provisioning request *requestPolicy()* (1) method is submitted by the maestro to its parent auctioneer to cover the additional resources as illustrated in Figure 5. The provisioning request enters the parent auctioning process, following either a successful admission control sequence by traversing the hierarchy tree a provider can accommodate the new demands. The concept of the virtual network hierarchy tree for resource management is novel part of virtuosity. An *AllocSpec()* (2) (see Figure 9 in Appendix) is returned back to the originating parent indicating the success of failure of the operation. The result is returned to the requesting child and if successful is stored in the *policy cache* as illustrated in Figure 5.

##### 3.3.2.2 Capacity Planning

The maestro can also leverage longer timescale measurements to proactively plan resource capacity using the parent's resource allocation system. By staying ahead of the demand growth of its local client base and child network demands, it can leverage capacity planning techniques widely used today to adjust capacity class provisioning demands. This results in better management of demand fluctuations, potentially reducing the number of times admission control needs to 'traverse' the virtual network hierarchy tree to accommodate new child virtual networks.

#### 3.3.2.3 Event-based Management

The maestro maintains global state information in distributed measurement cache. Parent and child network aggregate resource states and capacity class usage compiling profiles that summarized the state information over several timescales, e.g., per auctioning interval, minutes, and hours. The maestro leverages this measurement cache to influence distributed resource pricing used by the auctioneer when auctioning its resources to child virtual networks. A management event-based system allows parents to setup dynamic event invocations under conditions related to the admission of a new virtual network, resource usage inefficiencies (e.g., via the monitoring state, (4)), or renegotiation of parent resources (up or down) due to, for example, capacity class demand fluctuations. New events (*promptEvent()*, (5)) and the associated event specification (*EventSpec()*) will precede the auctioning process, whereby child virtual networks would then vie for resources. The IDL for the event specification is shown in Figure 9 in the Appendix.

#### 3.3.2.4 Slow Timescale Resource Allocation

The maestro invokes the auctioning process on a periodic or static deadline basis. This period is driven by slow timescale considerations, which we plan to investigate further through analysis, simulation and implementation but anticipate a duration in the order of tens of minutes, hence allowing the auctioning process to reach equilibrium and maintain constant services over longer timescales. The slow time scale is motivated by these factors, as is the virtual network creation timescale. While we believe that short-lived virtual networks will exist the vast majority of 'virtual network holding times' will be over longer durations. This indicates that virtuosity will only be invoked on timescales associated with virtual network creation and removal process, which will be far longer than today's call holding times.

Resource pricing and quantity announcements to child networks are set such that more effective utilization and revenue gain can be achieved by the parent. The maestro uses (6) two variables for resource auctioning. These are a price quote, $\boldsymbol{Q}_{ij}$, and a rate quote, $\boldsymbol{R}_{ij}$, (where $i$ = virtual resource; $j$ = capacity class) which the *delegate* element relays to the *seller object* of the resource allocation system for appropriate auctioning. The auctioning process (which is discussed in the next section) requires a recursive, distributed algorithm and global consensus in order to reach steady state [16]; this constrains the static or dynamic invocation process to a lower periodic bound for recurring resource allocations. Upon reaching auctioning equilibrium [16], the maestro receives the results (7) of the auctioning process, calculates local resource policies based on it's own resource policy that are set previously by its parent, and stores (9) the resource allocation policy results for child networks in the policy cache as illustrated in Figure 5.

#### 3.3.2.5 Policing

The maestro also cooperates with local delegates which distributes (10) resource allocations to each child virtual network, and then distributes local policy (*distributePolicy()* (11)) to the arbitrator for mediating and policing usage of the local parent resource by the child networks. Policing thresholds are set based on either capacity class allocations from the recent auctioning interval or from long-standing provisioning contracts (discussed in Sections 3.4). Monitors assume the responsibility of policing against these thresholds and noting violations appropriately in the measurement cache. Stored violation data can also be used by the maestro to inform or treat non-conforming child networks during the next provisioning interval.

### 3.4 Auctioneer

We propose a virtual network resource allocation process based on supply and demand of virtual network services where competing child virtual networks, working on behalf of a community of users and through appropriate specification, request resources and pay for such services to a provider of virtual network services. There are inherent behaviors and objectives that dictate the economics, and more importantly, the effective allocation, partitioning, and utilization of such services. We argue that the provider (parent virtual network) and subscriber (child virtual network) behaviors, and correspondingly their objectives serve as fundamentals that can be leveraged for resource maximization through the influence of economic variables. Network providers seek to achieve resource efficiency through the effective utilization of link resources through effective price-based, load balancing and the addition of multiple virtual network subscribers. Clearly, it is in the best interest of the parent to seek a greater number of subscribers for a given set of resources, without impacting their service requirements, in order to increase revenues. On the other hand, subscribers serve primarily the global interests of the virtual network users, but secondly seek to maximize the use of virtual network resources at a minimal cost. Finally, the virtual network users negotiate with the virtual network subscriber to request and achieve their required QoS. The competing nature that both the provider (parent) and subscriber (child) exhibit, we argue, should create the necessary dynamics that leads to a more aggressive environment for achieving resource efficiency.

### 3.4.1 Auction-based Resource Allocation

Auctioning of parent link resources is dealt with by sellers auctioning the parents *virtual links* capacity to competing child network buyer processes. Child virtual links are nested in the parent virtual links. Therefore, the process of auctioning the parents' virtual link capacity results in the child being allocated a percentage of the available capacity classes. As illustrated in Figure 6, the resource allocation process is coordinated through provisioning interfaces (*ReqSpec(); AllocSpec(); EventSpec()*) with the child maestros and the parent auctioneer; we described these in specific detail in later sections. During the spawning phase of the life-cycle, admission control is performed locally on parent resources to determine if the necessary communication, computation and memory resources are available to support the instantiation of a routelet for the newly requested child virtual network. As discussed in the previous section, the life-cycle spawning process coordinates with virtuosity to perform virtual network admission testing. Here bandwidth admission requires the parent virtuosity system to evaluate the request specification (*ReqSpec()*) for each required parent link resource to determine if the parent has sufficient bandwidth to support the provisioning request. Successful admission allows the request to proceed and compete through auctioning with other child virtual networks and vie for parent virtual network resources.

The auctioning process is decentralized to individual routelets where a seller object auctions partitioned bandwidth (i.e., capacity on an outgoing physical link[4]) to one or more buyer object bidding based on child virtual networks capacity class needs. The provisioning interface *ReqSpec()* operated between parent and child virtuosity systems (as illustrated in Figure 6) provides an open interface for capacity class-based provisioning of virtual networks. The rate and price specifications within the *ReqSpec()* are expressed for each required capacity class during a provisioning period.

#### 3.4.1.1 Long-Standing Auctions

Two important variables are the contract variables: contract_duration and contract_maxcost. These variables represent two important provisioning options which allow child subscribers to make long-standing contracts with the parent provider; in this sense, these variables represent a way for a child to avoid the normal open-market competition of the auctioneer. At a premium cost, subscribers can make long-standing contracts with the provider to avoid the dynamics or potential for provision 'bumping'. The contract acceptance criterion is based on a projected optimal investment value for the provider. The contract would require the parent network maestro to reduce the available and announced auction rate quantities to peer child network buyers, thus removing a portion of the auctioned resources for the duration for the contract.

### 3.4.2 Auctioneer Design

The auctioneer object architecture is illustrated in Figure 6. An auctioning agent interfaces with potential buyers and provisioning specifications. If a new 'buyer' is identified, a buyer object is created with its provisioning specification. Buyer objects seek to minimize cost and maximize rate quantity for any given capacity class. The seller receives optimal price and available resource quantities for individual capacity classes. This is dependent on what drives the provider market towards the desired revenue objective as well resource gain efficiency, e.g., preferring controlled capacity to constant capacity. The auctioning process is designed to follow a bidding procedure, allowing, for example, controlled capacity classes to be sold before constant capacity classes. In all cases, the auctioning of best-effort classes would follow the more stringent capacity classes. The per-class per-virtual resource bidding process may not reach a successful allocation conclusion for a particular buyer [16], if the buyer is unwilling to pay the market value for the resource capacity or does not offer alternatives. Alternative bids are structured based on ranges (e.g., percentage; rate_quantity*;* maxbid_payment) specified in the *ReqSpec()* and passed to the buyer for auctioning. Capacity class degradation strategies can be programmed where the auctioneer is unable to provide the first choice but may be able to accommodate a degraded alternative. This 'degraded request' is also identified by of the *ReqSpec()*.

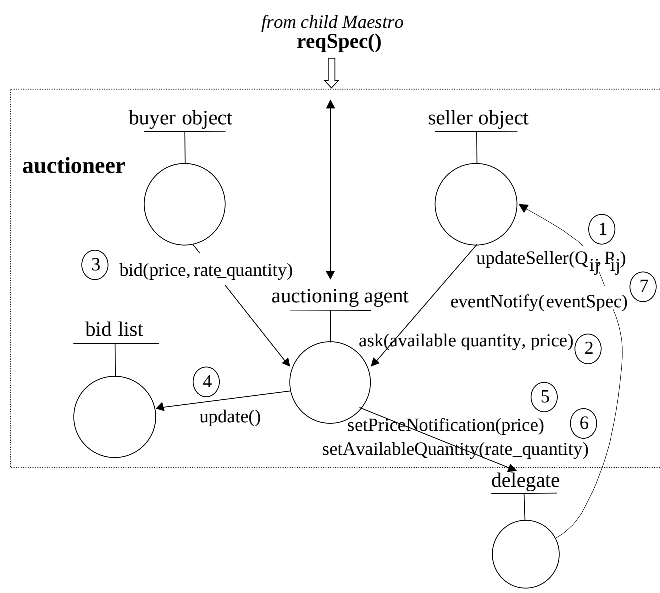


*Figure 6: Auctioneer Object Model*

---

4 Although we restrict the provider resource to the network link bandwidth, we feel that this model can be extended to support router resources or by partitioning router resources proportionally based on virtual network link aggregate demands.

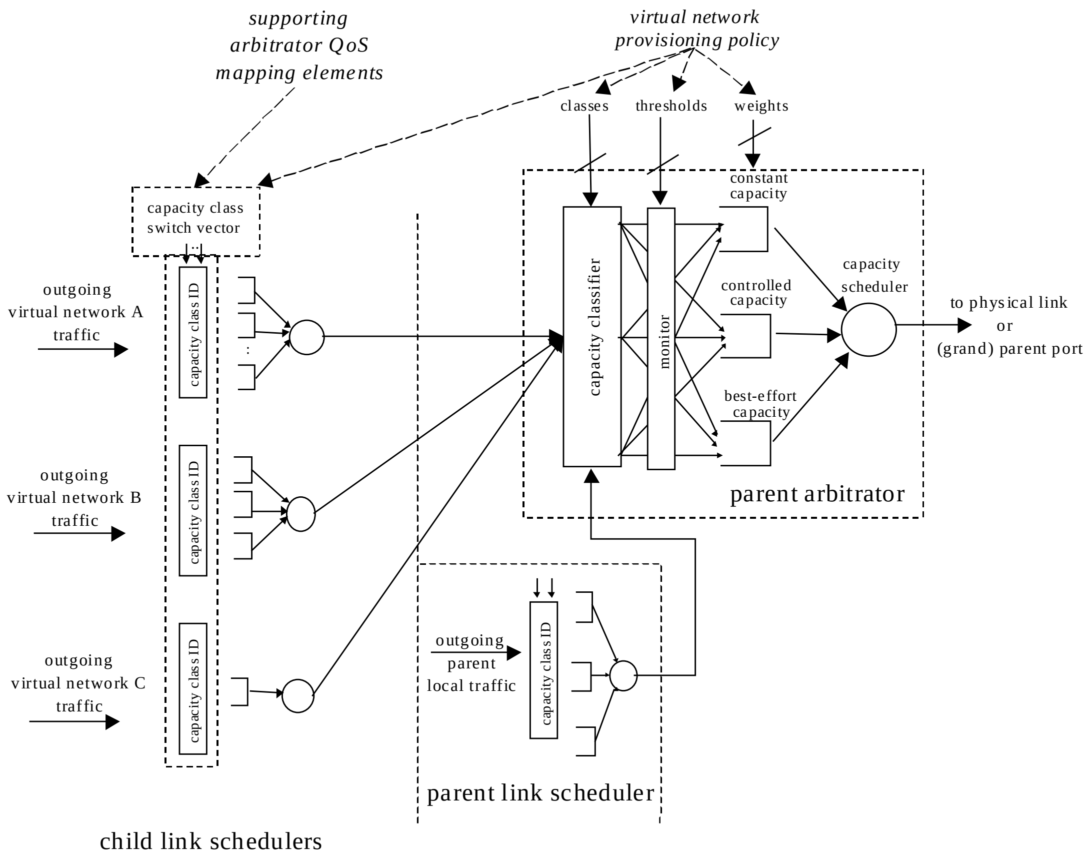


*Figure 7: Arbitrator*

#### 3.4.2.1 Auctioning Dynamics

The dynamics of the auctioning process is illustrated in step-wise form in Figure 6. A delegate, operating on behalf of the local maestro, coordinates event (1) invocations with the seller to notify the beginning of an auctioning process, and updates (2) the seller with optimal price and quantity strategies to influence the auctioning environment. The seller object then announces (3) the available resource quantities and associated pricing for buyer bidding. Following this, the auctioning agent, which maintains (4) a bid list, mediates the auctioning process between buyers and sellers seeking successful auctioning equilibrium and optimal resource allocations. Delegates communicate agreed price (5) and rate (6) allocations to the local maestro for child networks' policies for its provisioned share of the parent resource. The maestro then seeks closure by communicating the allocations (or denials) to child virtual network maestros through its delegates for final consideration. If agreed allocations are not satisfactory (e.g., a child network did not reach consensus on all requested parent resources and seeks greater share of the current resource) for any subscriber child network, the child maestro may invoke the auction process (again) with alternate specifications, and the previous child allocations and policies are voided by the parent maestro.

### 3.5 Arbitrator

A key component of virtuosity is its capacity scheduling ability. A virtual network scheduling abstraction resident at each parent resource arbitrates child virtual network access to the parent virtual links. The capacity arbitrator is based on a set of virtual network *capacity classes* and capacity *class weight* policies that are distributed to the arbitrator component by the delegate on behalf of the maestro. Capacity classes represent virtual network differentiated policy for provisioning capacity. Class weights are calculated (by the maestro) based on the following parameters: rate_allocation, percentage, price specified in the *AllocSpec()*. These policies are distributed on auctioneer provisioning boundaries upon completion of the resource allocation process. The capacity classes and weights translate the resource allocations negotiated by individual child virtual resources during the auctioning process to a set of virtual capacity scheduling policies. These policies are then used to differentiate child virtual network capacity allocations and the ordering of packet delivery to the parent link resource.

#### 3.5.1 Arbitrator Design

As discussed earlier, each child virtual network is assigned a unique virtual network ID to distinguish its traffic from other child network traffic. Also, by way of the spawning process, a *capacity class identifier* function is introduced into the arbitrator architecture prior to the child's link scheduler function to recognize QoS behavior treatment (e.g., best-effort, controlled load, expedited forwarding, etc.) associated with the child specific QoS architecture. This function interworks with a *capacity class switch vector* to assign each packet a *stamp* that associates it with a particular capacity class, prior to its arrival at the child's link scheduler as illustrated in Figure 7. A switch vector is formed from the required capacity classes requested during the provisioning process and allocated during the resource allocation process to the child virtual network customer. If, for example, a customer or child network supports only best-effort IP traffic classes within a spawned child network and provisions for constant capacity, the switch vector would stamp all traffic with a constant capacity classification. On the other hand, if the customer supports Integrated Services classes (viz. guaranteed, controlled load and best-effort) within a spawned child network and provisions for all three capacity classes, then the class identifier would stamp the traffic with corresponding capacity classifications, by default.

#### 3.5.2 Packet Treatment

The introduction of the virtuosity arbitrator into the output port architecture merges both child and parent QoS scheduled traffic and provides coarse capacity scheduling of the composite traffic based on the allocated provisioning policies. A capacity classifier is used to identify virtual networks and their capacity classes. The classifier queues incoming stamped packets (from the output of child network link schedulers) to the appropriate capacity queue structures (viz. constant, controlled, and best-effort). Individual queues are created for each child virtual network within an allocated capacity queue structure. Each virtual network queue is then assigned an appropriate weight, based on the policy previously negotiated and distributed by the maestro. Within the provisioned interval, the arbitrator manages scheduling of virtual network control based on the capacity class priority and weights, allowing child networks (and local user traffic) to queue available packets to the parents output link scheduler. The capacity arbitrator leverages space (i.e., available resource bandwidth), time (i.e., provisioning interval length) and capacity-class abstractions to manage scheduling of its own user traffic and packets from child virtual networks onto the parent link scheduling facility. The capacity arbitrator services the capacity queues in priority order and weighted round-robin for same capacity class queues.

It is important to note that the illustration represents the default virtual network resource management implementation and is *not* the only parent option for managing child network traffic. Alternatively, the parent may override the arbitrator function and integrate child network traffic through its local routelet port link scheduler. The architectural selection and realization is based on the parent resource management policy set during the profiling phase and composed during the spawning phase of the life cycle process.

## 4 Implementation Considerations

### 4.1 Timescale

The issue of stability of the virtuosity resource management system is conditioned by the programmed policy. It is therefore important that policy-based dynamic provisioning promotes the stability of the network. We believe that by limiting the timescales over which dynamic resource provision operates, we can achieve a balance between the gains derived from statistical sharing of resources between virtual networks and the desired stability of virtual networks. There are several considerations that provide a trade-off in stability and resource efficiency. First, network services which operate within the context of virtual networks, should operate at some defined steady state rather than continuously fluctuate. Second, provisioning intervals must be lower-limited such that admission control and auctioning processes for resource allocation must reach convergence within minutes. Moreover, the infrequent requirement for child networks to do dynamic provisioning will keep the virtuosity system fairly inactive, and so, a relaxation on the lower bound provisioning interval should not detract from resource efficiency objectives. Finally, the timescale trade-off will be influenced further by virtual network node size and hierarchy tree depth complexity. The issue of timescales in relation to the proposed framework will be investigated in the implementation phase.

### 4.2 Distributed Design

Key operational considerations in the development of the virtuosity framework are scalability and performance; both of which are important factors to any good distributed design strategy. One of the obvious benefits of the spawning inheritance model, is the fact that we can reduce state and space complexity. We observe several design considerations associated with resource management architecture for managing virtual networks:

- network complexity (e.g., virtuosity management element interactivity (i.e., traffic overhead) and virtual network size);
- computation complexity (e.g., routelet management processing overhead);
- transport data path impact; and
- frequency of control/management interactions.

In our proposal we have selected to use the maestro and delegate strategy to centralize management intelligence and processing (e.g., displaced to a centrally configured policy server), yet decentralize the interactive activity required between the delegate and the other virtuosity components. Also, by selecting a single delegate model per-routelet, we can scale management processing with node complexity rather than link resource complexity.

The auctioning system could be designed based on a per network, per resource or per-routelet basis. Such a design choice should consider auctioning equilibrium convergence speed [16] and processing overhead in support of multiple auctioning processes. Currently, we favor a per-routelet auctioneer strategy, which is potentially more scalable given the slow timescales required to support dynamic provisioning. Finally, we require an arbitrator for each parent link. An issue associated with arbitration is classification of a virtual network's traffic. In this case we plan to use a VN-ID and capacity class stamp allowing virtuosity to manage multiple child virtual networks' traffic without inspecting the packet.

## 5 Conclusion

The creation, deployment and management of network architecture is manual, time consuming and costly. In this paper, we have presented an overview of spawning networks and the virtual network kernel being developed within the Genesis Project [9] at Columbia University. The virtual network kernel is capable of architecting, "spawning" and managing distinct virtual network architecture. The primary contribution of this paper is the definition of virtuosity, a framework for virtual network resource management, which is an integral part of the Genesis virtual network kernel. The status of our work is as follows. We have defined a spawning architecture and its virtuosity framework for virtual network creation and management, respectively. We are currently building spawning networks through the implementation of the virtual network kernel and its virtuosity plug-in.

## 6 Acknowledgement

This work is supported in part by the National Science Foundation (NSF) under CAREER Award ANI-9876299 and with support from COMET Group industrial sponsors. In particular, we would like to thank the Intel Corporation, Hitachi Limited and Nortel Networks for supporting the Genesis Project. The authors would like to thank Michael Kounavis, Nemo Semret and Raymond Liao of the COMET Group for their insights and helpful discussions. John Vicente (Intel Corp) would like to thank the Intel Research Council for their support during his visit with the Center for Telecommunications Research, Columbia University. Daniel A. Villela would like to thank the National Council for Scientific and Technological Development (CNPq-Brazil) for sponsoring his scholarship at Columbia University (ref. 200168/98-3).

## 7 References

[1] Biswas, J., et al., "Application Programming Interfaces for Networks", IEEE P1520 Working Group Draft White Paper, www.ieee-pin.org/

[2] Blake, S., et al. "A Framework for Differentiated Services", draft-ietf-diffserv-framework-01.txt.

[3] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss W., "An architecture for differentiated services", draft-ietf-diffserv-arch-02.txt, October 1998.

[4] Session on "Enabling Virtual Networking", Organizer and Chair: Andrew T. Campbell, OPENSIG '98 Workshop on Open Signaling for ATM, Internet and Mobile Networks, Toronto, October 5-6 1998.

[5] Campbell, A.T., De Meer, H., Kounavis, M.E., Miki, K., Vicente, J., and Villela, D. A., "A Survey of Programmable Networks", Computer Communications Review, April 1999.

[6] Campbell, A. T., De Meer, H. G., Kounavis, M. E., Miki, K., Vicente, J., Villela, D. A., "The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures", IEEE 2nd International Conference on Open Architectures and Network Programmability (OPENARCH'99), October 1998, pp. 115-127.

[7] DARPA Active Network Program, www.darpa.mil/ito/research/anets /projects.html, 1996.

[8] Duffield N., et al., "A Performance Oriented Service Interface for Virtual Private Networks", draft-duffield-vpn-qos-framework-00.txt. Work in progress.

[9] The Genesis Project: Programmable Virtual Networking http://comet.columbia.edu/genesis, 1998.

[10] Keshav, S., and Sharma, R., "Achieving Quality of Service through Network Performance Management", Proc. of NOSSDAV'98, Cambridge, July 1998.

[11] Lazar,A.A., "Programming Telecommunication Networks", IEEE Network, vol.11, no.5, September/October 1997.

[12] Lazar, A.A. and A.T Campbell, "Spawning Network Architecture", White Paper, Center for Telecommunications Research, Columbia University, http://comet.columbia.edu/genesis, Janurary 1998.

[13] Van der Merwe,J.E. and Leslie,I.M., "Switchlets and Dynamic Virtual ATM Networks", Proc Integrated Network Management V, May 1997.

[14] Van der Merwe,J.E., Rooney,S., Leslie,I.M. and Crosby,S.A., "The Tempest - A Practical Framework for Network Programmability", IEEE Network, November 1997.

[15] Multiservice Switching Forum (MSF), www.msforum.org

[16] Semret, N., and Lazar, A. A., "Design, Analysis and Simulation of the Progressive Second Price Auction for Network Bandwidth Sharing", Technical Report CU/CTR/TR 487-98-21

[17] OPENSIG Working Group comet.columbia.edu/opensig/

[18] Rajan, R., Martin, J. C., Kamat, S., See, M., Chaudhury, R., Verma, D., Powers, G., Yavatkar, R., "Schema for Differentiated Services and Integrated Services in Networks", draft-rajan-policy-qosschema-00.txt, October 1998. Work in progress.

[19] Rooney, S., Van der Merwe, J. E., Crosby, S. A., Leslie, I. M., "The Tempest: A Framework for Safe, Resource-Assured, Programmable Networks", IEEE Communications Magazine, October 1998, pp 42-53.

[20] Touch,J. and Hotz,S., "The X-Bone", Third Global Internet Mini-Conference in conjunction with Globecom '98 Sydney, Australia, November 1998.

[21] Valko, A. G., Campbell, A. T., Gomez, J., "Cellular IP", INTERNET-DRAFT, draft-valko-cellularip-00.txt

[22] Ferguson, P. and Huston, G., "What is a VPN?", OPENSIG'98 Workshop on Open Signalling for ATM, Internet and Mobile Networks, Toronto, October 1998.

# Appendix

```
interface Auctioneer{
        void ReqSpec(in short i, in long VN_ID, in ConstantCapacityClass constCapClass,
                in ControlledCapacityClass contCapClass,
                in BestEffortCapacityClass bestEffCapClass,
                inout long contract_duration, in long contract_ maxcost)
        raises (Reject);
        // i is the request order, i=0 is primary, i=1 is the alternate or degraded request spec, j=2, etc.
        // VN_ID is the VN identification; VN-0 is the root physical network
        // the classes are defined as structs containing related parameters
        // contract_duration is the commitment duration request for a future reservation
        // contract_maxcost is the maximum payment for the future reservation

};
```

*Figure 8: CORBA IDL ReqSpec() for the Auctioneer*

```
interface Maestro{
        void AllocSpec(in long VN_ID, in ConstantCapacityClass constCapClass,
                in ControlledCapacityClass contCapClass,
                in BestEffortCapacityClass bestEffCapClass, in long contract_ID)
        raises(Reject);
        // VN_ID is the virtual network Identification
        // the classes are defined as structs containing related parameters
        // contract_ID, if not Null, allocation is part of a new or long-standing contract

        void EventSpec(inout long VN_ID, inout short event_class, inout short event_type)
        raises(Reject);
        // VN_ID virtual network prompting the event
        // event_class either parent or child dynamic invocation or parent static deadline
        // event_type, applies to the dynamic case: renegotiation, new VPN, resource availability,
                // capacity upgrade, downgrade, contract negotiation, etc
};
```

*Figure 9: CORBA IDL Event() and AllocSpec() for the Maestro*