Programming Internet Quality of Service

John Vicente[1, 2], Michael Kounavis[1], Daniel Villela[1],
Michah Lerner[3], and Andrew Campbell[1]

[1] Center for Telecommunications Research, Columbia University, NY, USA
{mk, dvillela, jvicente, campbell}@comet.columbia.edu

[2]Intel Corporation, USA
john.vicente@intel.com

[3]AT&T Labs, USA
michah@att.com

Abstract. The deployment for new Internet services is limited by existing service creation platforms which can be characterized as closed, vertical and best effort in nature. We believe there is a need to develop a programmable Internet built on a foundation of open service interfaces and middleware technologies. To help speed the introduction of value-added services, we propose a unified, programmable Quality of Service (QoS) API framework based on the IEEE P1520 Reference Model fostering open, standard interfaces for networks. We argue that this is a necessary evolutionary step towards a QoS-flexible, Internet service platform. We propose the design of APIs for upper level network QoS be based on service-dependent and service-independent abstractions, supporting alternative styles of QoS specifications and provisioning. Additionally, we propose the design of low-level network element APIs be based on the notion of a building block hierarchy and the separation of service-specific and resource abstractions for the creation and deployment of network services.

1 Introduction

The increasing value of e-business, seen for example in home markets and personal financing, impels the Internet toward universal connectivity supporting diverse end-user requirements. Businesses compete for improved customer access and greater productivity. This fuels the development of personalized services and interactive environments including "virtual offices". Reliable and pervasive services may leverage the competitive innovations of multiple providers, and this requires flexibility, service differentiation, isolation, privacy, and manageability. The basic IP infrastructure provides, however, a limited service-delivery platform for the introduction of new scalable network services. More than "best effort" data transport, these services increasingly require guaranteed Quality of Service (QoS) with pervasive security. Some customers may require Service Level Agreements (SLA) as well. Reliable and simple realization of such goals may soon leverage ubiquitous directory technologies placing identities and information on an open network. The control and usage of such information is essential as the Internet supports increasingly complex activities. In turn, this creates demand for policy-based management at multiple layers from transport to content. Coordination of authentication, authorization and accounting appears

C. Linnhoff-Popien and H.-G. Hegering (Eds.): USM 2000, LNCS 1890, pp. 106-123, 2000.

mandatory, and these need to function over diverse standards and technologies. This guides the Internet towards a service-delivery platform, thus satisfying the increasingly diverse and demanding customer requirements.

The commercial success of the scalable Internet infrastructure may be limited unless the constituent entities support interoperability while preserving differentiation of service quality; this may exploit dynamic configuration as well as management of fluctuating demands. Convergent solutions of these challenges occur within the baseline of "stateless best effort" IP (ISO layer three). This may leverage recent protocols (DiffServ [1], IntServ/RSVP [2,3] and draw upon diverse transport capabilities. Existing solutions for IP-based service quality utilize alternative transport composed of multiple layers (e.g., ATM, frame relay, MPLS [4]).

These vertically deployed technologies may induce an unintended and dangerous partitioning of the Internet into multiple and incompatible network domains. This occurs as an artifact of using proprietary architectures with distinct QoS semantics. Thus, various solutions are being proposed which deal with traffic class translation [5], network integration, and virtualization techniques [6]. Indeed, the diverse protocols reflect the multiple transport requirements. Interoperability between these protocols may occur through Application Programming Interfaces (APIs) as adaptation layers.

While progress has been made toward network QoS programmability [7], [8], [9], there has been little work to unify QoS programmability across architectural platforms and distinct network QoS domains. In this paper, we propose a set of QoS APIs spanning multiple network programmability layers to support application, network service and device-level QoS programmability essential to the formation of an Internet service-delivery platform. Such uniform APIs define the essential *protocol invariant* specification of QoS. We propose a unified QoS API framework for programming Internet QoS based on the IEEE P1520 Reference Model [10]. Rather than assume a single API can satisfy the diverse requirements at multiple transport layers, P1520 describes interactions between standard building blocks.

This unified QoS framework provides sets of low-level APIs. These comprise the resource-specific and service-centric QoS abstractions. Each level encapsulates the notion of building blocks that enable network device programmability. At the platform level, these APIs act as layered services that insulate the end-user from the complexity of network algorithms (e.g., admission control, reservations, or service level agreements). These support programming interfaces suitable to a network-domain's specific QoS characteristics. In addition, a higher-level and architecture-independent interface establishes the QoS of Internet sessions. New Internet applications leverage this interface without need for detailed resource-provisioning knowledge of the network domains. The adapter objects convert architecture-independent interfaces to architecture-dependent interfaces. These adapters may also maintain policies for provisioning, accounting, charging and billing.

The paper proceeds as follows. In Section 2 we discuss the emergence of programmable networks. We introduce our programmable QoS API framework in Section 3. In Section 4 and 5, we present and discuss the QoS API design forming the lower and upper QoS interfaces, respectively. Following this, we discuss related work in Section 6. Finally, in Section 7, we present some concluding remarks.

2 Programmable Networks

2.1 Open Signaling and Active Networking

The current developments in network programmability have mainly sourced from the OPENSIG (Open signaling) [11] and DARPA Active Networks [12] communities. The OPENSIG approach argues for a set of open, programmable network interfaces, enabling access to internal state and control of network devices (e.g., switches, routers). Thus independent software vendors (ISV) and service providers can enter the telecommunications software market, thereby fostering the development of new and distinct architectures and services. Alternatively, the DARPA Active Network community, operating mainly within the IP communication model has offered a more radical approach to network programmability by advocating dynamic deployment of new services at runtime. While both communities have made good progress towards the development of programmable networks, much of this progress has taken on a research for "new architectures" motivation. The Internet industry, on the other hand, has been slow in adopting programmable networks within the commercial marketplace. Moreover, current network systems consist of vertical, proprietary architectures, duplicating the mainframe orientation of bundled OS, hardware and underlying services. Therefore, we believe a necessary step to achieving success or evolution in the Internet is the "horizontalism" of network systems through standardization of common, open APIs.

2.2 Programming Interfaces for Networks

The IEEE P1520 standardization effort is leading industry initiatives for standard programming interfaces for networks (PIN) to enable rapid service creation and open signaling. The scope of P1520 activities covers technology from ATM switches to IP routers and media gateways. The P1520 Reference Model (RM), illustrated in Figure 1, is a generalized model for network programmability structured without architectural or technology context. The interfaces, as shown, support a layered design offering services to the layers above it while abstracting the components below it for customization or programming. Each level comprises a number of *entities* in the form of algorithms or objects representing logical or physical resources depending on the level's scope and functionality.

More specifically, we distinguish the interfaces as follows:

- User level programming interfaces, collectively called the *V interface*, provide access to the value-added services level. This interface provides a rich set of APIs supporting highly personalized end user applications through value added services.
- *U interfaces* exist between the value-added services level and network generic services level (NGSL). The U interface deals with generic network services. The power of this interface comes from its generality, creating a separation between the actual interface and vendor implementations, allowing multiple network-level schemes to coexist in a single network.

- The *L interface* provides programming interfaces between the network generic services level and the virtual network devices level (VNDL). The L interface defines the API to directly access and manipulate local device network resource states.
- Open protocols to access the state of physical elements. Collectively called the Connection Control and Management interface, the *CCM interface* is a collection of protocols that enable the exchange of state and control information at a very low level between a device and an external agent.

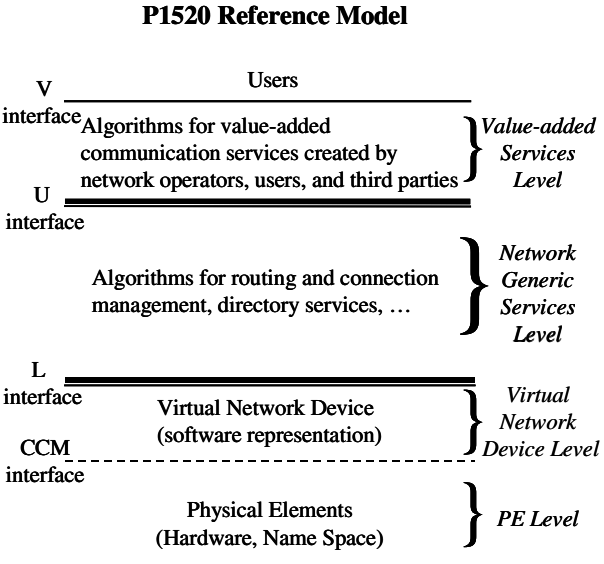

Fig. 1. IEEE P1520 Reference Model

2.3 Internet Service Platform

The P1520 RM provides a framework for positioning programming interfaces and operations of networks over a specific networking technology. A contribution of this paper is a proposed mapping of current Internet-based QoS technologies and their corresponding functionality to the P1520 RM. This enables customization to support valued-added services by use of the underlying network infrastructure for the delivery of diverse services and QoS requirements. This customization is essential to a malleable service environment that adapts to new services or requirements. Ensuring this customization is the assumption that the network does not impose *a priori* limits upon the diversity of providers of networks, services, content or other functions.

The deployment of QoS on the scalable Internet is inherently complex due to the range of traffic classifications, numerous providers, and service levels. This complexity should not burden the network, nor induce network partitions due to loading variations. Sustained reliability, scalability and capacity should not become vulnerable to mandatory access and migration of state information, and should remain relatively

insensitive to temporary outages or unannounced reconfiguration. While one approach push substantial state information to the network edges, such an approach may also be perceived as shifting the state management to less informed or capable exterior elements. Such evolving issues present somewhat of a challenge to the ongoing Internet growth. Thus, simplification through common and open APIs is an essential element of the solution. Deployments of such standard interfaces may either manage or delegate the state information as the technologies allow. This applies the well-understood software engineering principle of standard interfaces to common components. Various API implementations are largely interchangeable provided they satisfy the interface and functional requirements.

API deployments often exploit adaptive middleware technologies. In particular, a service-oriented middleware structure supports the highest layer of the QoS framework, through definition of services and QoS at the points of service definition and service invocation. Although few practical examples are available in the literature, the AT&T GeoPlex project [13] established the concept of an open, service-oriented platform operating over IP supporting a multitude of complex and heterogeneous Internet services [14]. The GeoPlex adhered to unifying design principles thereby imbuing services with a stable behavior. It demonstrated sophisticated facilities such as coordination through usage recording and asynchronous event processing; administration through account and profile management; managed security including firewall-enforced authentication, sign-on and access control; as well as network integration including proxies, protocol translation and mediation. Using middleware and middleware services, GeoPlex provided a common management structure including global account hierarchy (with reference to network directories as available) with uniform APIs. These APIs allow adaptation to multiple system management solutions, each with their particular advantages.

These common APIs isolate functionalities and avoid unnecessary implementation constraints. Control and functionality may therefore move to the most effective locations. In particular, this migrates network control (often called “intelligence”) closer to the core networking resources, and enables effective use of high bandwidth resources. This is compatible with emerging network architectures. We view the integration of QoS with the provisioning infrastructure without such as a middleware platform, analogous to having resource APIs within an computer platform architecture without the operating system services necessary to manage, isolate or protect critical resources. Few people would argue that the OS services are unnecessary complexity. However, within the context of a “networked” platform the complexity extends to include network protocol heterogeneity, mobility, address management, security policy integration and finally, inter-domain SLA coordination. To illustrate appropriate alignment, we anticipate the integration of QoS APIs with an open service platform, e.g., [13], [14] at the equivalent V interface of the P1520 Reference Model.

3 A Framework for Programming Internet QoS

The motivation for network QoS includes optimal service delivery, service differentiation and network bandwidth efficiency. The motivation for network programmability includes flexibility, customization, reconfigurability and portability.

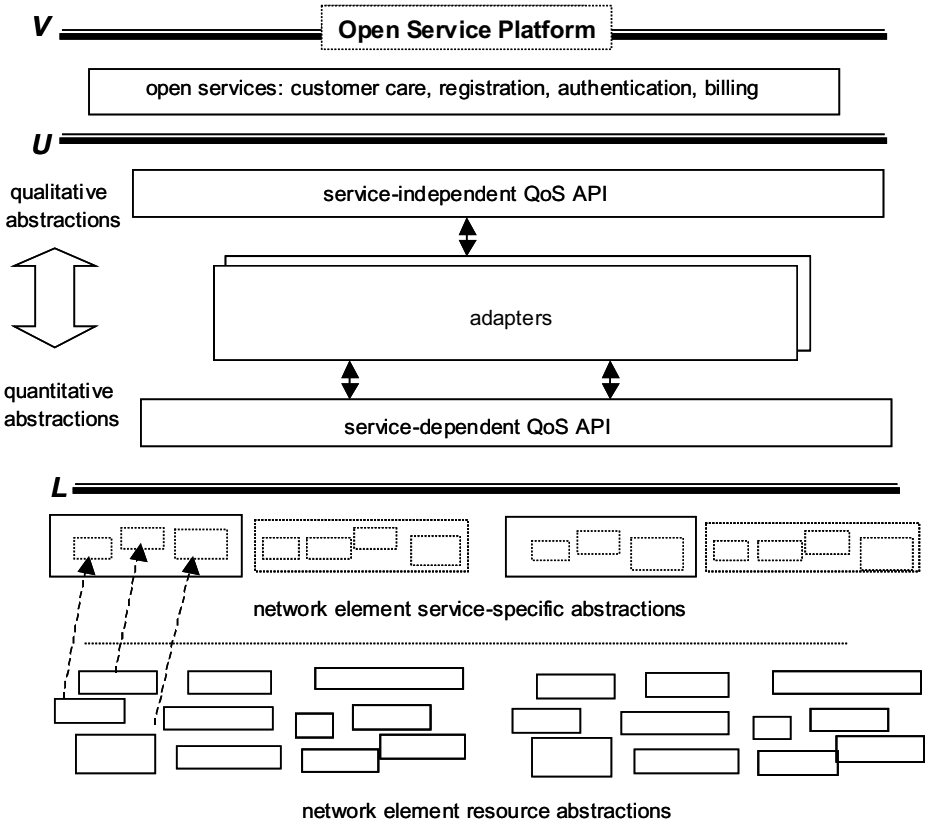

Fig. 2. Programmable Internet QoS Model

As we define programmable QoS, it is the notion of having network systems that enable customization and flexible integration of QoS over multiple layers of the communication model. Whether it is the seamless integration across application layers, or network devices, we advocate open interfaces that would minimize complexity and heterogeneity of network infrastructures to an appropriate higher layer abstraction. The P1520 Reference Model inherits the architectural hierarchy of value-added services, network services and algorithms, device and protocol implementations in its layered structure.

We propose a unified QoS API framework to support application, network service and device-level QoS programmability essential to the formation of an Internet *service-delivery* platform. More specifically, our contribution focuses on U (upper) and L (lower) interfaces of the P1520 Reference Model as a unified approach to deliver programmable QoS. Figure 2 depicts a conceptual view of the QoS API framework integrated with a general open platform API structure and superimposed onto the P1520 RM. We describe the architectural perspective of our proposal in the next sections, and provide specific detail in Section 4 and 5, corresponding to the design of the L and U interfaces, respectively.

3.1 Network Element QoS API

In pursuing a common design of QoS interfaces for network elements, there are fundamental requirements that we must adhere to. First, we recognize that there is a clear requirement for network providers to differentiate their proprietary product features and implementation, such that market competitiveness is preserved through product or service delivery superiority including performance, design features, availability/reliability, cost efficiency, etc. Complementary to this, is the clear requirement to enable service provisioning by ISVs and ISPs to foster innovation and rapid service deployment within the spectrum of Internet network services. So, a necessary balance must be struck, as we design network element APIs such that both requirements are satisfied. Secondly, with increasing intelligence being designed into network elements, we anticipate traditional packet and flow processing becoming the default network behavior, while emerging devices will perform multiple functions thereby defining a new class of network elements that extend behavioral functionality within the network transport. Thus, the traditional routers and switches are subsumed with multi-service aggregation devices (e.g., VPN) with explicit multi-function requirements. This will include address translation, firewall enforcement, proxy activation, load balancing, and advanced monitoring.

Therefore, to meet these challenges, we have proposed an API structure that is based on abstraction granularity and scope. More specifically, the first order of abstraction is the separation of the *service-specific* and *resource abstractions*. The second order of abstraction is the notion of "building block" abstractions, which are course resource abstractions that can be combined to form *i.)* default behaviors within a network element (e.g., scheduling); *ii.)* new resource-level behaviors; or finally, *iii.)* service-level behaviors supporting service-specific abstractions. As in [15], we propose exposing control, management and transport functionality associated with the objects by way of the semantics of the interface definition. This includes the discovery of available resource abstractions, their composition, configuration, policies and state. In summary, the proposed L interface is formed from fundamental object-oriented design principles with the goal towards the development of open, extensible network element APIs.

3.2 Network Services QoS API

There are many commonalities which exist between alternative approaches for guaranteeing QoS in multimedia networks today. For example, the Differentiated Services Framework defines the notion of a 'leased line emulation' service, a service closely associated with flows in the Integrated Services Architecture, and virtual circuits in ATM networks. However, the QoS, requirements associated with each - leased line emulation services, flows and virtual circuits - are widely disparate. For example, the quantitative descriptions for QoS of multimedia sessions differ significantly between alternative networking technologies. Although all descriptions attempt to codify the predictable statistical characteristics of multimedia streams, the traffic assumptions associated with these descriptions are not the same.

The QoS mechanisms (i.e., admission control, scheduling, reservations) used by a particular network domain to support service differentiation and resource control are reflected in the service model characterizing that domain. While appropriate U interfaces should capture the granularity in which service models allow the specification

of QoS for multimedia sessions (e.g., the flowspec in the IntServ model), alternatively U interfaces must be generalized in a manner that allows seamless development and integration of Internet applications and middleware platforms. These two design requirements contradict each other. To satisfy both, we separated the quantitative from the qualitative descriptions of QoS requirements in the U interfaces, as illustrated in Figure 2.

Therefore, what we propose is to have a *service-dependent* layer that is open and extensible offering quantitative and network-specific abstractions for multimedia services, and a *service-independent* layer of abstractions offering more qualitative and application-specific abstractions for multimedia applications. The lower-level abstractions are rigidly defined for different technologies and hide the complexity of different network algorithms. The upper-level abstractions are loosely defined and span multiple service-specific technologies, hiding the heterogeneity of the service models associated with the different network domains. By 'loosely defined', we mean that some information, necessary to establish sessions with QoS, is concealed at this level. Such information is handled by an 'adapter' object, which also performs the necessary conversion and mediation between the layers to meet the requirements of each. Adapter objects can be introduced dynamically into end-systems and in the network using a variety of methods. Adapter objects are specific to user preferences, network domain characteristics, and the types of services offered. Under an open service platform, as described earlier, QoS services can dynamically generate adapter objects on behalf of their clients, when the clients register with the platform. The design of the U interfaces and the role of adapter objects is further explained in the Section 5, where we also present the design of service-dependent interfaces for the Integrated Services and Differentiated Services architectures.

4 Open Network Elements

4.1 Virtual Network Device Level Interfaces

As illustrated in Figure 3 below, the proposed L interface is formed of three models of abstraction. This is an architectural extension from our previous work on the two-layer model [15]. Similar to [15], our approach enables network device programmability from two complementary perspectives, corresponding to an association with the layers of the L abstraction model, primarily service-specific and resource. This allows, for example, upper level interfaces to create or program completely new network services using resource abstractions or modify existing services using service-specific abstractions, which are themselves built on resource abstractions. The third layer is introduced to facilitate common device programmability via a standard set of base building block abstractions, which both the service-specific and resource layers are built.

Furthermore, the abstractions within these three layers differ in form to the previous work [15], as we propose an aggregation model based on building blocks. The upper part of the L interface is the service-specific abstraction layer of the network-element. The *service-specific building block* abstractions at this layer expose "sub"-interfaces associated with underlying behaviors or functions, state or policies on the local node

that have concrete meaning within the context of a particular supported service (e.g. Differentiated Services). The idea here is that an administrator or ISV need only program the device within the context of the service rather than deal with low-level abstractions associated with generic resources (e.g. scheduler, dropper) of the network device. Therefore, in order to deliver the required service-specific behavior, the programmer need only modify or provision the service abstraction at the level that they understand or have a need (or privilege) to supplement.

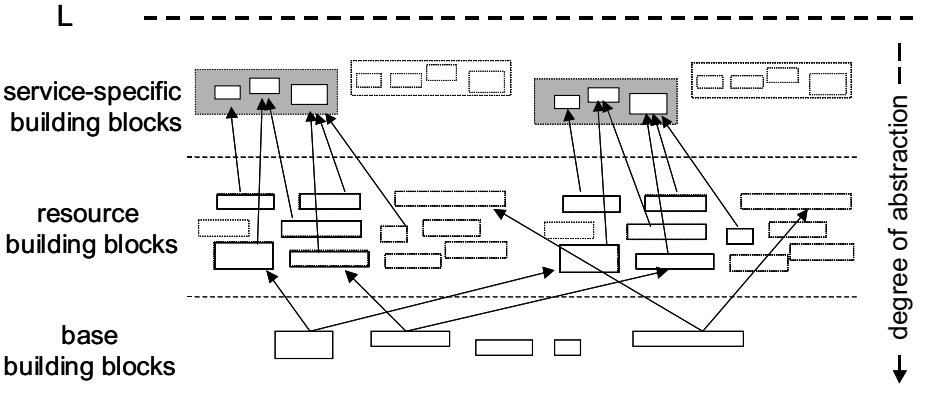

Fig. 3. P1520 L interface abstraction model

Alternatively, the middle part of the L interface abstraction model is the resource abstraction layer of the network element. The abstractions here are termed *resource building blocks*, from which default behaviors (e.g., Diffserv PHB) or new behaviors can be built. We envision the programmer as a sophisticated ISV developer or network software architect, who is knowledgeable of underlying resource abstractions of a network element[1] (e.g., router) and can construct new behaviors or functions, and can change state or policies within the context of the abstraction without knowledge of the underlying vendor device implementation.

The maximum degree of abstraction is achieved at the lowest layer of the abstraction model. This new layer is fundamental to the design of the proposed L interface. The idea behind the *base building blocks* are to have abstractions that have no service or resource significance from a network element behavioral or packet processing perspective. These base blocks serve the needs of the programmer, only in an inheritance fashion such that abstractions above the base layer (namely resource or service-specific) can be designed appropriately to create new functional service behaviors or resources or modify (enhance) existing ones in a consistent, standard object-oriented manner.

[1] We use the term 'network element' loosely here, as we position L interfaces such that resource building block or service-specific abstractions need not be associated with just a router device, but any network device that processes network flows or IP packets.

4.2 Designing Network Element QoS Interfaces

Rather than trying to align standardization across proprietary resource or service implementations, we propose a model for API development and standardization that can be built on common building blocks. Furthermore, the building block concept allows varying degrees of abstractions for network element programmability. For example, one can support either a "queue" building block or "traffic conditioner" building block that can be a collection of subcomponents including shapers, droppers, etc, which form the traffic conditioner. The API standardization discussion becomes a matter of resources or services that are deemed sufficiently common and open for service provisioning and do not compromise any compliant (e.g., P1520) vendor's proprietary features. The proposed model makes the L interface truly open, flexible, reusable, extensible and dynamic.

Table 1. Base building blocks

Building block	Variable	Methods	Description
Action	ID, priority, type	discover () actionmethod () target () raises (exception)	Performs a specific function on the directed path from which the processing unit traverses within the network element.
Component	ID, status, type	discover (target()) get(target()), set(target()), delete(target()), add(target()), modify(target()) target () internal-block(target()) raises (exception)	A resource or component entity on the router. This may be a singular component (e.g., queue), or an aggregate component consisting of several building blocks.
Condition	ID type	predicate () target () raises(exception)	A decision point, possibly pre-staging an action or component.
Target	ID, type	reference(nextblock) raises(exception)	Provides a reference to the next building block, from which the previous entity directs the network element processing.
Processing Unit (PU)	ID, state, size		This is the unit of processing through the router, i.e. a packet.

4.2.1 Base Building Blocks

As listed in Table 1, the base building blocks provide the foundation to the building block concept, and have architectural significance in serving the needs of the programmer through inheritance of these base classes. Thus, as new resource or service-specific abstractions are created, they may leverage the inherited characteristics offered by the base classes. Building of new service-specific behaviors, resources or resource functions can be accomplished through the inheritance of base classes, and the concatenation or chaining of one or more service-specific or resource building blocks

and their corresponding methods. The programming and deployment models are considered out of scope of this document, but can range from rule-based approaches (e.g., policy-based management), active networks language-based techniques to quasi-static techniques (e.g. CORBA).

4.2.2 Resource Building Blocks

The resource building blocks represent the functional entities of the network element that either perform action (e.g., mark packet) or function; exist as generic resources (e.g, buffer or queue) or guide decisions (e.g., route table) based on some criteria. The granularity of abstraction is a key question with regard to standardization; as exposed control or manipulation of critical network element resources may compromise the vendor's proprietary features that allow the vendor to differentiate itself in terms of scalability, performance or service availability.

Table 2. Example resource abstractions

Abstraction	Methods	Description	Derived from
TrafficClass	SetTrafficClass() GetTrafficClass() GetTrafficClassID()	It contains the traffic classes defined for the programmable element. Traffic classed are defined by QoS and traffic description.	Component
Filter	SetType() GetType()	Implements a packet filter. More complex filters are derived based on the type of filter. Examples are the BA filter and the MF filter.	Action
Classifier	SetFilter() GetFilterInfo()	Classifiers map traffic based on a set of filters to different streams in a 1:N relationship. More complex classifiers can be defined as well.	Action
Meter	SetType() AddProfileEntry() ChangeProfileEntry() DeleteProfileEntry() SetNonConforming() GetMeterStatus ()	Measures the level of conformance of the traffic to specified profiles.	Action
Scheduler	GetAlgorithm() SetAlgorithm()	Implements the scheduling function and can be defined upon a queuing discipline.	Action
Queue	SetPriority() GetPriority() SetMaxProfile () GetMaxProfile()	Stores the packets at some stage on the programmable element. It can receive different parameters such as priority, guaranteed rate etc.	Component
Profile	Set(AvgRate, Delta, Gain, BurstSize) Get(AvgRate, Delta, Gain, BurstSize)	The profile is defined in terms of rate, burstiness, the amount of traffic (in bytes) in a small fixed sampling interval (delta) etc.	Component

Thus, a degree of granularity and the proper selection of resources, which necessitate ubiquitous operation are key to the question of API standardization. Table 2 below provides some typical examples of resource building blocks within a router or network element, along with their associated base abstractions from which they inherit fundamental building block characteristics.

4.2.3 Service-Specific Building Blocks

The level of abstraction achieved at this layer serves the needs of the ISP, ISV or application developer by facilitating service provisioning, service control and management from a pre-defined service perspective. For example, under DiffServ, a DSProvision class can be created as a DiffServ service-specific building block. It may be formed on underlying resource building blocks (e.g., traffic class, profile, meter, queue, scheduler) or it may inherit the component and condition building blocks and their characteristics to form a simple output port (e.g. on DSCP=xx, *set(target(local))*). Therefore it is possible to build services over existing resources building blocks or directly from the base abstractions. The flexibility provides the programmer more or less control of the resources that it depends on to accomplish the task.

Table 3. *Example* - Diffserv service-specific abstractions

Abstraction	Methods	Description	Derived from
DSTrafficClass	setType() getType() setDSTrafficClass() getDSTrafficClass()	Implements a Differentiated Service traffic class based on traffic description and QoS description.	TrafficClass
DSProfile	SetDSProfile() modifyDSProfile() deleteDSProfile()	Contains specification of a Differentiated Service profile.	Profile
DSMonitor	getStats()	Collects measurement information on the DiffServRouter	Monitor
DSProvision	getQoS() setQoS()	Receives information about desired QoS, e.g., average delay, packet loss rate etc.	Provision

5 Open Network Services

5.1 Network Generic Service Level Interfaces

Programmable networks enable the composition of distributed network algorithms over network element interfaces. Programmable networks should also enable the deployment of higher-level middleware platforms without exposing the complexity of the distributed algorithms characterizing their architectures. Moreover, developers should be provided with simple, open interfaces for programming and controlling network services. In response to these requirements, the P1520 reference model specifies the U interface as a Network Generic Service Level interface for modeling the functionality of the network. Communication algorithms can control open programmable routers/switches by invoking methods supported at the L interfaces, while exposing U interfaces to higher-level entities. The U interface captures the functionality of resource allocation, admission control, routing, and network management algorithms, providing standard means for using any network architecture. As illustrated in Figure 4 below, U interfaces are grouped into three categories:

- *Transport interfaces* are used for controlling the transport protocol stacks, which are operational within a network domain. Examples of transport interfaces include

monitoring and protocol stack management interfaces. Monitoring interfaces are used for obtaining feedback about the quality of established sessions within a network domain (e.g., a video conferencing session). Protocol stack management interfaces can be used for dynamically composing or modifying transport protocol stacks on behalf of middleware users.

- *Network control interfaces* are used for managing communication resources among competing parties in the system, as well as for accessing or modifying routing information. Network control interfaces support higher level communication services with Quality of Service guarantees. Resources can be allocated to support point-to-point, point-to-multipoint sessions, or they can be used for dynamically creating virtual private networks.
- *Network management interfaces* support open management functions such as policy-based management, network monitoring, event management or capacity planning.

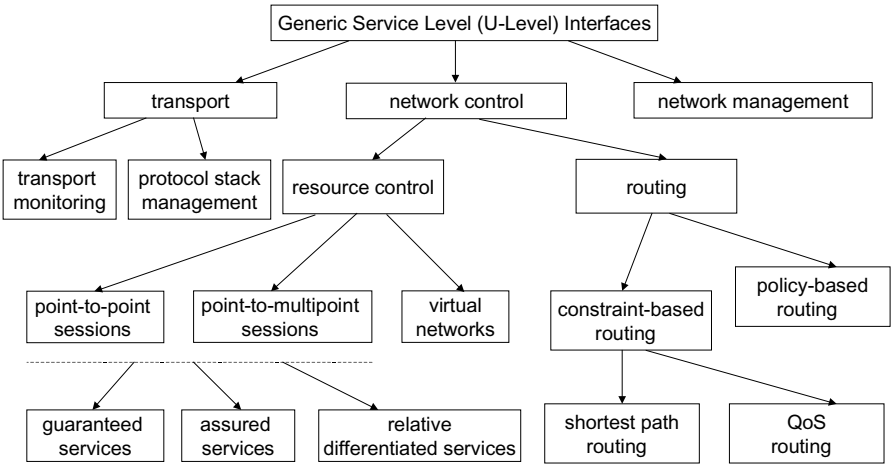

Fig. 4. Family of U interface

5.2 Designing Service-Level QoS Interfaces

U-level interfaces represent the macroscopic counterparts of functions supported by network elements through their L interfaces. The main difference between the L and the U interfaces is that L interfaces have local significance only (i.e., abstract the operations of a single network node), whereas the U interfaces abstract distributed services operating across one or multiple network domains. We believe that there is a need to understand the level of abstractions and scope characterizing the U interfaces to develop more flexible, efficient and customizable service platforms in the next generation Internet. Among the family of the U interfaces, discussed above, resource control interfaces are the least understood. While different network architectures support alternative mechanisms for providing QoS to their users, it is difficult to specify a generic QoS API which can characterize the underlying QoS architecture and semantics of every network domain.

5.2.1 Service-Independent Interfaces

The design of the service-independent abstraction layer is motivated by the commonalities, which exist between traffic classes of different network architectures. Most network architectures support some type of service with very low or bounded queuing delay. Examples of such service include the guaranteed service in Integrated Service networks and the leased line emulation service in Differentiated Service networks. In addition these architectures support the notion of a service that is better than best effort, but does not provide explicit delay guarantees. Controlled load and assured forwarding-based services belong to this category. A third group of services comprises relative differentiated services.

Table 4. Service-level QoS interfaces

Abstraction	Methods	Description
GenericResourceController	ReqGuaranteedService() RemQuaranteedService() ReqAssuredService() RemAssuredService() ReqRelativeService() RemRelativeService()	A generic QoS abstraction that allows the establishment or removal of multimedia sessions between a pair or group of end-systems with some QoS assurances.
IntServResourceController	ReqGuaranteedService() RemGuaranteedService() ReqControlledLoad() RemControlledLoad()	A QoS abstraction layer specific to the Integrated Service (IntServ) resource model. This layer allows the establishment or removal of multimedia sessions, which are characterized by the guaranteed service and controlled load classes of the IntServ model.
DiffServResourceController	ReqLeasedLineService() RemLeasedLineService() ReqAssuredService() RemAssuredService() ReqOlympicService() RemOlympicService()	A QoS abtraction layer specific to the Differentiated Service (DiffServ) resource model. The DiffSDerv architecture aims in the standardization of per hop behaviors, not end-to-end traffic classes.

The service-independent layer supports the division of services among guaranteed, assured, and relative differentiated services. This distinction is qualitative, not quantitative. The quantitative characteristics of different traffic classes are concealed at this level. Table 4 lists the `GenericResourceController` interface with corresponding methods that support the generic QoS specifications. Methods supported at this service-independent layer accept two generic abstraction parameters - namely `rate` and `application_type`. These parameters are passed into adapter objects, which convert the service-independent interfaces to appropriate QoS 'service-specific' interfaces. As described earlier, adapters capture QoS provisioning intelligence including user preferences and service provider policies.

5.2.2 Service-Dependent Interfaces

As shown in Table 4, the `IntServResourceController` interface abstracts the QoS provisioning characteristics of Integrated Service networks. Integrated Service networks support two traffic classes with QoS assurances: guaranteed service and controlled load service. The guaranteed service ensures that datagrams associated with an Internet session arrive within some specified delay bound. To support guaranteed service, routers need to schedule link capacity among competing flows. In addition, resource reservations need to be setup along the path which datagrams follow. A controlled load service tightly approximates the behavior visible to applications receiving best-effort service under unloaded conditions. Network elements support controlled load service by allocating service rate and buffer space, which are larger than the token bucket rate and size characterizing the multimedia source. The difference between the guaranteed and controlled load services is that in the controlled load case no explicit rate allocations are requested. The rate allocated to multimedia flows depends on the implementation of network elements and it is not always high enough to guarantee that queuing delays are below a deterministic bound.

Integrated Services networks have been associated with scalability problems, due to the requirement for maintaining per-flow state in network elements. To overcome this deficiency the IETF is currently pursuing stateless resource management using a differentiated service model. In differentiated services networks user flows are compared against negotiated traffic profiles before entering a network domain. Packets are marked with code-points corresponding to specific forwarding functions. A small group of standardized forwarding functions, called Per-Hop Behaviors (PHBs), are used for realizing higher level services that do not require per-flow state management.

As described in Table 4, the `DiffServResourceController` interface abstracts the QoS provisioning characteristics of Differentiated Service networks. Three distinct types of differentiated service are supported. *Leased line emulation* service ensures the delivery of customer traffic with very low latency and very low drop probability. Leased line emulation service uses an 'expedited forwarding' per hop behavior. *Expedited forwarding* per hop behavior has the highest priority. *Assured service* is used for carrying data traffic at higher priority than competing best effort traffic. Supporting the corresponding methods in Table 4, the sustained traffic rate characterizing a leased line emulation service and the offered rate which characterizes an assured service are specified in a `LeasedLineSpec` and `AssuredServiceSpec` data structures (not shown), respectively.

Both the leased line emulation service and the assured service support absolute traffic profiles. A fundamentally different approach in the differentiated service framework is the relative differentiated services. In this approach the quality spacing between different traffic classes is not defined deterministically. Relative service differentiation does not require admission control for the establishment of service level agreements. Relative differentiation can be based on strict prioritization, price differentiation, or capacity differentiation. To accommodate this, we specify corresponding methods in the interface for an 'olympic' service, a generic relative differentiated service. In this case, user traffic and differentiation (e.g., cost or otherwise) is handled according to a `traffic_class` parameter.

6 Related Work

New developments in the area of open networks or network programmability have emerged with a sense of realization. This has been motivated by research progress[2] founded mostly from the DARPA formed Active Networks community and OPENSIG initiatives. Several projects [16], [17][3], [9], [18], [19] have demonstrated unique contributions towards the development of programmable network control and QoS. Alternatively, very little activity from the IETF has focused on standardizing APIs. However, we are encouraged by the following recent developments:

- The formation of the Multiservice Switching Forum [20] is committed to an open, standardized control interface that will support multiple control planes.
- The Parlay [21] consortium, paralleling similar ideas of an open service platform, provides an extensible API specification that includes a framework and service interface for customer care (e.g., registration, security, billing, management, etc) and value-added services (e.g., call control, messaging), respectively.
- Complementary to Parlay, the Java Advanced Intelligent Networks [22} effort, focusing on the convergence of the telephony and data networks, provides a middleware framework to enable service creation and service execution. This is achieved by way of a specification of component abstractions for call control, coordination and transaction models that are independent of implementation specific call models, protocols and underlying infrastructure.

In general, there has been very little published material from these initiatives that address open programmability for Internet QoS -- addressing flexible QoS service creation, provisioning and SLA delivery across heterogeneous network domains and infrastructure equipment. Alternatively, consider the several recent IETF initiatives that have similar motivations to the above and been proposed towards interoperability and/or the development of management-based QoS:

- The IETF has established a GSMP [23] Working Group that is focused on developing a general switch management and control protocol for switching devices.
- In [24], the authors propose a SNMP MIB-based interface standard for Differentiated Services QoS. Similar to our work on the L interface, the work here is heavily influenced by [25] and takes a rigorous approach to modeling lower level resources of the router to support slow timescale control, provisioning, monitoring and event management.
- Similarly, the authors in [26] provide a management interface model (i.e., to [23]) to support the provisioning of QoS, using policy and policy-based semantics through COPS (Common Open Policy Services) [27]. The QoS Policy Information Base (PIB) also abstracts network devices providing the means to set policies for resource definition and configuration, provisioning, and discovery of underlying QoS resource abstractions.

---

[2] See [29] for a more complete survey of the activities here.

[3] Both xbind and Tempest initiatives have spun off new companies -- xbind Inc. and Cplane, respectively.

- A contribution by Hitachi, LTD [28] leverages the SNMP protocol to create a programming interface MIB for routers. Here, the work is close to ours as QoS programmability is introduced through the concept of a "virtual flow labels" (VFL) to form modular policy rules, and extending static conditional semantics such as those used in [24].

While the industry and research communities have very different agenda and timelines, we view our contribution addressing a middle ground and practical step towards the evolution of the Internet agenda for open networks and flexible, unified QoS control and delivery

7 Conclusion

In this paper we have presented a programmable QoS API framework based on the IEEE P1520 Reference Model. We have discussed the QoS API framework in the context of network service and network element quality of service deployment, and suggested its integration with a open service platform which provides the necessary infrastructure to support QoS inter-workings over the complex and evolving Internet. We have discussed the design of APIs for upper level network QoS, which is based on a service-dependent and service independent model; supporting alternative styles of QoS specifications and provisioning. Similarly, we presented an abstraction for designing underlying network elements based on the notion of building blocks and the separation of service-specific and resource context for network element programmability.

References

[1] Y. Bernet, S. Blake, J. Binder, M. Carlson, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, W. Weiss, "A Framework for Differentiated Services," Internet-Draft draft-ietf-diffserv-framework-01.txt, work in progress, Feb. 1999.

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP)," Version 1 Functional Specification, RFC 2205, Sept. 1997.

[3] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, June 1994.

[4] Xiao Z, Ni L.M., "Internet QoS: Big Picture," Dept of CS, Michigan State University.

[5] Andrikopoulos I., Pavlou G., "Supporting Differentiated Services in MPLS Networks," Proc. 7th International Workshop on Quality of Service (IWQOS'99), London, May 1999.

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

[7] Adam, C. .M., et al., "The Binding Interface Base Specification Revision 2.0", OPENSIG Workshop, Cambridge, UK, April 1997.

[8] Angin,O., Campbell,A.T., Kounavis,M.E. and Liao,R.R.-F., "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking", IEEE Personal Communications Magazine, Special Issue on Adaptive Mobile Systems, August 1998.

[9] Chandra, P. et al., "Darwin: Customizable Resource Management for Value-added Network Services", Sixth IEEE International Conference on Network Protocols (ICNP'98), Austin, October 1998.

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

[11] OPENSIG Working Group http://comet.columbia.edu/opensig/

[12] DARPA Active Network Program, http://www.darpa.mil/ito/research/anets /projects.html,

[13] Vanecek, G., Mihai, N., Vidovic, N., and Vrsalovic, D., Enabling Hybrid Services in Emerging Data Networks, IEEE Communications Magazine, July 1999.

[14] Lerner, M., Vanecek, G., Vidovic, N., and Vrsalovic, D., Middleware Networks: Concept, Design and Deployment of Internet Infrastructure; Kluwer Academic Press, April 2000 (ISBN 07923-78490-7)

[15] Denazis, S., Vicente J., Miki K., Campbell A., “Designing Interfaces for Open Programmable Routers”, July. 1999.

[16] Lazar, A.A., Lim, K.S. and Marconcini, F., ``Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture," IEEE Journal on Selected Areas in Communications, Special Issue on Distributed Multimedia Systems, No. 7, September 1996.

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

[18] Raguparan, M., Biswas, J., Weiguo, W., L + Interface for routers that support Differentiated Services, May 1999.

[19] Vicente, J., Biswas, J., Kounavis, M., Villela, D., Lerner, M., Yoshizawa, S., Denazis S., A Proposal for IP L Interface Architecture, January 2000.

[20] Multiservice Switching Forum (MSF), http://www.msforum.org/

[21] The Parlay Group, http://www.parlay.org/

[22]Java Advanced Intelligent Network (JAIN)
http://java.sun.com/aboutJava/communityprocess/jsr/jsr_018_oam.html

[23] General Switch Management Protocol from Peter Newman (Netsim Inc www.ensim.com/) find the specification at http://www.iprg.nokia.com/protocols/rfc1987.htm

[24] Baker, F. et al, “Management Information Base for the Differentiated Services Architecture” Internet Draft, draft-ietf-diffserv-mib-01.txt. Oct. 1999.

[25] Bernet, Y., Smith, A., Blake, S., A Conceptual Model for Diffserv Routers, October 1999, INTERNET DRAFT.

[26] Fine M., McCloghrie K., Hahn S., Chan K., Smith A., "An Initial Quality of Service Policy Information Base", draft-mfine-cops-pib-01.txt, Internet Draft, June 1999.

[27] Boyle J., Cohen R., Durham D., Herzog S., Rajan R., Sastry A., “The COPS (Common Open Policy Service) Protocol,” Internet Draft draft- ietf-rap-cops-07.txt. Aug. 1999.

[28] Kanada Y. et al, “SNMP-based QoS Programming Interface MIB for Routers” Internet Draft, draft-kanada-diffserv-qospifmib-00.txt., Oct. 1999.

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