A Framework for Resource Negotiation and Pricing in the Internet

Xin Wang, Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
xwang@ctr.columbia.edu, schulzrinne@cs.columbia.edu

Abstract

Network delivery services providing "better-than-best-effort" service over the Internet are particularly necessary for multimedia applications. The selection and use of a specific delivery service involves negotiation between the user and the network; they agree upon specifications such as the type of service user packets will receive, the constraints the user traffic must adhere to, and the price to be charged for the service. In this paper, we describe a protocol through which the user and the network (or two network domains) can negotiate network services. We refer to the protocol as a Resource Negotiation and Pricing protocol (RNAP). Through RNAP, the network service provider communicates availability of services and delivers price quotations and charging information to the user, and the user requests or re-negotiates services with desired specifications for one or more flows. RNAP protocol mechanisms are flexible enough to support multiple delivery service models, and allow dynamic re-negotiation of services during a session. Two different network architectures are defined to support RNAP, a centralized architecture with a Network Resource Negotiator (NRN) administering each network domain, and a distributed architecture without any centralized controlling entity. Mechanisms are proposed for local price and charge computation, formulation of end-to-end prices and charges across multiple domains, and communication of this information through RNAP messages. Results of a prototype implementation are described briefly.

1 Introduction

Motivated by growth of Internet multimedia applications, a number of researchers have investigated network delivery services that provide "better-than-best-effort" (BBE) service to the user, in the sense that they provide some QoS guarantees to applications. Important examples of proposed network service models are the integrated service model (int-serv) [1, 2], and the differentiated service model (diff-serv) [6, 7].

As these services are implemented in the Internet, user applications will be able to request and use the delivery service appropriate to their requirements. We may regard the selection and use of a specific delivery service as a negotiation process. The customer and network negotiate and agree upon specifications such as the type of service user packets will receive, the constraints the user traffic must adhere to, and the price to be charged for the service. The central goal of our work is to develop a protocol through which this negotiation can take place. The protocol should be generic and flexible enough to support multiple delivery services and environments (including int-serv, diff-serv, and best effort services), service negotiation at different levels of granularity (flow- and aggregate-based), negotiation by both sender and receiver, and "in-band" and "out-of-band" resource reservation mechanisms. It should allow the service provider to communicate service availability, estimated prices for available services and charges accruing to the user, and allow the user to request a specific service. It should also support dynamic service re-negotiation between the user and the network, allowing the network to adjust pricing in response to changes in network load, and allowing the user to respond to changes in application requirements. We refer to the proposed negotiation protocol as the Resource Negotiation And Pricing protocol (RNAP).

Based on the policy of each domain, different algorithms can be used for computation of a local or incremental price for a service at a given point in a network; We propose a number of alternative mechanisms to allow the network to compute a global price on the basis of these incremental prices, and to charge the user for the end-to-end service. The proposed protocol and architecture and pricing mechanism are intended to co-exist with current Internet QOS schemes (e.g. those proposed within the int-serv and diff-serv frameworks), and work in a scalable manner over a variety of network architectures. We present RNAP as a stand-alone protocol, but it is also possible to implement some components of RNAP as a layer on top of RSVP or other hop-by-hop reliable signaling protocols.

RNAP is intended for use by both adaptive and non-adaptive applications. Non-adaptive applications may choose services that offer a static price, or absorb any changes in price while maintaining their sending rate. Adaptive applications adapt their sending rate and/or choice of network services in response to changes in network service prices. RNAP provides a framework within which an application can adapt so as to obtain the best value from the network.

The paper is organized as follows. In the following section, we define two alternative protocol architectures, a centralized architecture (RNAP-C), and a distributed architecture (RNAP-D). We consider the design goals discussed above in greater detail in order to develop an outline of the RNAP protocol. In Section 3, we present a detailed description of the basic RNAP protocol, including protocol messages, and message sequences in centralized and distributed architectures. We then expand our discussion to end-to-end service negotiation across multiple domains, and also briefly consider advance reservation mechanisms. In Section 4, we discuss pricing and charging mechanisms in RNAP. The communication of pricing and charging information in the various RNAP messages is first discussed, followed by an explanation of how end-to-end pricing and charging can be formulated under both RNAP-C and RNAP-D architectures. We end the section by considering a specific strategy for pricing a BBE service at a single network point, based on which a complete pricing system may be realized using RNAP. In Section 5, we describe a prototype implementation of the RNAP protocol and architecture in a test-bed network. In Section 6, we briefly discuss some related work. We summarize our work in Section 7 and also point out important open issues.

2 Architecture and Design Goals

2.1 Protocol Architecture

We begin by considering a scenario in which a customer (sender or receiver) wishes to reserve network resources for multiple flows, for example, traffic flows from a video-conference. We assume that the user application negotiates through an agent referred to as the Host Resource Negotiator (HRN). The HRN is responsible for obtaining information and price quotations for available services from the network. During the negotiation, the HRN requests a particular service, specifying the type of service (guaranteed service, control load service, premium service, assured service, best effort service, etc.), and parameters to characterize the requested service. Some parameters are general to all services (immediate/advance reservation, preemption level, partial reservation, etc.) and other parameters are specific to a service class (peak rate, average rate, burst size, lost rate, delay, jitter etc.). The HRN can negotiate simultaneously for one or multiple flows, and request different services for each of them.

A HRN negotiates only with its access network to reserve resources, even if its flows traverse multiple domains. If a domain could provide pricing information for services along different paths, the HRN will choose the optimal path at beginning of the transmission. A HRN may also decide to renegotiate resources at a later time if the network is under heavy congestion and the price is prohibitive. In addition to resource

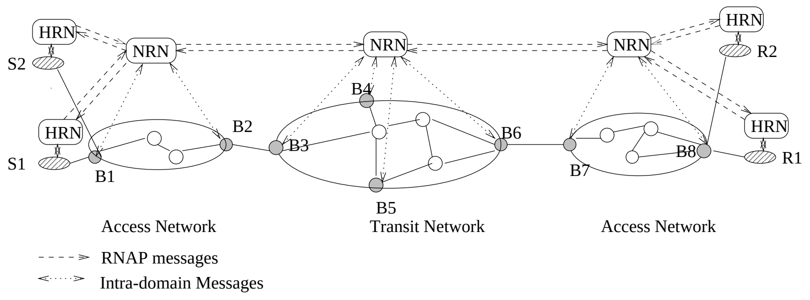


Figure 1: RNAP-C Architecture

negotiation between the HRN and the network, the RNAP protocol is also intended for resource negotiation between two network domains. An access domain "A" may receive requests for a service in a certain direction passing through a neighboring transit domain "B" from one or more users, and use RNAP to request the service for the flow or flow-aggregate from domain "B". We discuss an end-to-end negotiation scenario across multiple domains in Section 3.4.

For negotiation on the network side, we consider two alternative architectures, a centralized architecture, and a distributed architecture.

2.1.1 Centralized Architecture (RNAP-C)

In a centralized architecture, the network negotiates through a Network Resource Negotiator (NRN). Each administrative domain has at least one NRN. The NRN delivers price quotations for the different available service levels to HRNs, answers service requests from HRNs, and is also responsible for maintaining and communicating user charges for a particular session.

The NRN may be an individual entity, or may be a complementary functional unit that works with other administrative entities. For example, the NRN can be part of (or function as) the Bandwidth Broker (BB) in the diff-serv model [6] and the PDP in the COPS architecture [30]. The NRN either has a well-known address, or is located via the service location protocol [37]. The NRN address of a neighboring domain can be pre-configured or obtained through DNS SRV [38].

Resource reservation and admission decisions may be performed by the NRN; they may also be performed by other entities, such as the BB of the diff-serv model. If they are performed by other entities, the NRN communicates requests for services to them individually or in aggregate, and receives admission decisions and possibly pricing decisions from them. The implementation of resource reservation and admission control, and the associated communication with administrative entities, is closely related to specific BBE services, and is outside the scope of the RNAP protocol.

2.1.2 Distributed Architecture (RNAP-D)

In this architecture, networks don't have centralized negotiating entity. Instead, the protocol is implemented at routers in the network, and RNAP messages propagate hop-by-hop, from the first-hop router to the egress router, and vice-versa. We consider the messaging process in greater detail after introducing specific RNAP messages in Section 3.3.

Evidently, the RNAP-D architecture has in-band messaging, whereas the RNAP-C architecture has out-of-band messaging.

The RNAP message format is independent of the architecture. Therefore, the two architectures can co-exist; for instance, a domain administered by a NRN can exchange RNAP messages with a neighboring

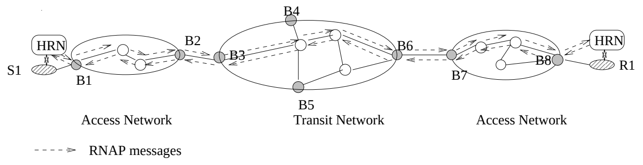


Figure 2: RNAP-D Architecture

domain which employs the distributed architecture. Also, a HRN does not need to know about the RNAP architecture of its local domain, since it receives and sends the same negotiation messages in either case.

2.2 Dynamic Re-negotiation Capability

There are a number of reasons that make it desirable for the negotiation protocol to permit services to be re-negotiated dynamically. In general, the network would like applications to acquire network resources so that there is high network utilization, but not at the expense of poor QoS. The real time constraints of multimedia traffic make it difficult for these applications to estimate the bandwidth required for an application.

Also, many existing multimedia applications allow the media rate and quality to be adjusted over a wide range, allowing them to respond to network congestion by gracefully reducing their rate [9], possibly utilizing application-specific knowledge. Such applications have the incentive to re-negotiate a service with lower QoS when network congestion results in the current service becoming more expensive, or if the network provider denies the requested service because of unavailability of the amount of resources requested.

Possible re-negotiation scenarios include periodic re-negotiation, in which the service contract expires after a period and is re-negotiated, and asynchronous re-negotiation initiated either by the customer or by the network provider. The RNAP protocol uses both mechanisms. Each service has an associated *Negotiation Interval*, during which the negotiated price and service characteristics remain constant. The resource reservation expires after the negotiation interval, so in order to maintain uninterrupted service, the HRN needs to re-negotiate the resource reservation request periodically. To facilitate the re-negotiation process, the network periodically sends the HRN service price and availability information. The periodic re-negotiation mechanism is optional, and a HRN not willing to negotiate may disable the mechanism at any time. The periodic re-negotiation mechanism allows the network provider to use network resources more efficiently, and also convey to the users the network state through pricing information (for example, when congestion occurs, the network tries to reduce the traffic entering the network by increasing the price). When the network is congested, an user capable of dynamically adjusting its transmission is able to respond to increase in price by adjusting its quality of transmission gracefully. Alternatively, the user can maintain a high quality of transmission by paying a higher price. It is likely that a negotiation with longer interval carries a “risk premium” to protect against network dynamics.

2.3 Pricing and Charging Capability

A network service model that provides one or more “better-than-best-effort” delivery services must also incorporate a pricing system, so that users are charged appropriately for different levels of service. Researchers have also suggested usage/congestion sensitive pricing as a congestion-control mechanism [12, 13, 14, 15, 25, 17, **?**, 18] if applications are capable of adaptation, as discussed above.

A pricing system includes monitoring of user traffic, price formulation at one or more points within the network, computation of a global, or end-to-end, price for a particular service, and a mechanism to communicate pricing information from the network to the customer. We consider these issues in Section 4. For the present, we assume the existence of mechanisms which enable the network (the NRN in RNAP-C, and individual routers in RNAP-D) to compute the price for a service, and to compute charges accruing to the user for services used. The RNAP protocol provides the means to communicate to the customer price quotations for different services, and the charge for services provided to the customer. It also supports different charging modes: charging the sender, or receiver, or both. The periodic re-negotiation framework provides a natural way to communicate periodic price quotations and cumulative charges to the customer.

2.4 Scalability

RNAP messaging is scalable in the sense that message volume is independent of the hop count of a route or the number of transit domains on the route. Scalability is therefore determined mainly by the need to transmit, maintain and process the state information relating to each multimedia session (consisting of one or multiple flows) established by a HRN. It is likely that individual customer flows will be progressively aggregated to form larger granularity flows in the core of the network. The NRN (or boundary routers of a domain in RNAP-D) may negotiate resources for such a flow, consisting of traffic belonging to more than one customer, entering from a neighboring network. In this case, the NRN or network does not have knowledge of individual flows belonging to the aggregate, and only maintains RNAP state information for the aggregation. The centralized architecture has better scalability, since the state information needs only to be maintained by the NRN and boundary routers of a domain.

2.5 Service Predictability

In general, each particular delivery service model has associated mechanisms to assure that the service received by the user is predictable. Predictability includes the quality expected from a service type, and the price charged for it. The periodic price quotation mechanism discussed earlier can also serve to increase the predictability of the overall service by keeping the price constant during a negotiation period.

2.6 Transport Protocol and Reliability

RNAP messages are sent using the UDP protocol. In both RNAP-C and RNAP-D models, synchronous RNAP messages are sent periodically and provide a natural way of protecting against loss. Since a negotiation involves charging, a HRN may want to know the current service price before sending out a new request. RNAP allows the HRN to asynchronously solicit any service related information at any time during the negotiation session. If a reservation request is lost in transmission, the network will continue to provide service based on contracted rules from previous negotiation period. If one or more *Reserve* messages are lost in transmission, the network continues to provide service based on the latest *Reserve* for a specified time-out period.

The HRN also sends messages asynchronously. To protect against asynchronous message loss (and as an additional protection against synchronous message loss), the HRN continues to retransmit a request with exponential back-off (for congestion control) until a response is received. The retransmission interval starts at close to the end-to-end round trip time. The retransmission interval doubles after each packet transmission.

Network failures, such as failure of a negotiation server or of a device storing RNAP state information, and network partitions need also to be considered. A back-up NRN may be needed for the RNAP-C model. If there is any possibility of resource unavailability due to element failure or route change, a re-negotiation process is triggered from the influenced domain to the corresponding neighboring domains to allow new resource agreements to be reached. When a device that stores customer charging information is down for a period, the charge for the period is asynchronously retrieved after the device becomes alive. The accumulated charge may need to be stored in a non-volatile storage.

The network should be able to track the liveness of an application using RNAP by tracking periodic RNAP messages and also by monitoring the flow. This would avoid charging a terminated application and wasting network resources.

2.7 Security

RNAP messages can be authenticated and encrypted in the same way as RSVP [39]. Alternatively, IPSEC [40] may be used.

3 Basic Negotiation Protocol

In this section, we start with an explanation of some basic terminology used in describing protocol messages, followed by a description of the protocol messages, and the typical negotiation sequence in which they are used. In the discussion that follows, we assume for convenience the RNAP-C architecture, and refer to the NRN as one of the negotiating entities. We later extend the discussion to the RNAP-D architecture, with the routers along the delivery flow path collectively playing the role of the NRN. Also for convenience, we assume that the other negotiating entity is a HRN, acting on behalf of the user application. As mentioned earlier, the RNAP protocol is also applicable to resource negotiation between two network domains, in which case, the first domain (through its NRN, in case RNAP-C is employed) plays the role of the HRN.

3.1 Terminology

Id: The *Id* field contains 3 sub-fields: *Flow Id*, *Aggregation Flag*, and *Aggregate Flow Id*. The*Flow Id* defines a flow for which services are negotiated. For individual flows, the *Flow Id* contains the source IP address and port, destination IP address and port, and optionally, the transport protocol. It also contains the destination network address, determined by the HRN by a method such as RADB lookup [44]. The purpose of the *Aggregation Flag* and *Aggregate Flow Id* sub-fields is explained when we discuss message aggregation, in Section 3.5.

Service: The *Service* identifier defines the service being negotiated. The HRN uses it to request a price quotation or reserve resources for a particular service with a set of associated parameters. The NRN uses it in the corresponding acknowledgment messages. A service identifier consists of *Service Type*, *Service Independent Parameters (SIP)* and *Service Specific Parameters (SSP)*.

SIP specifies a list of parameters that are generic to all service models and used to characterize a service. The service independent parameters include *Service Type*, *Starting Time*, *Ending Time*, *Negotiation Capability*, *Negotiation Interval*, *Preemption Capability*, and *Reservation Coverage*.

Service Type: The *Service Type* identifies a service. Examples of services are the Guaranteed and Controlled Load service models defined within the int-serv framework [1, 2], and the Premium and Assured Service models defined within the diff-serv framework [6, 7].

Starting Time and Ending Time: *Starting Time* and *Ending Time* specify the time period over which service is requested (when specified by HRN) or is available (when specified by NRN). The *Starting Time* and *Ending Time fields* are optional and may be used to make either immediate or advance reservations (Section **??**).

Negotiation Capability: The *Negotiation Capability* flag is used by the HRN to signal its capability or willingness to negotiate during session initiation.

Negotiation Interval: The *Negotiation Interval* defines the length of time over which the negotiated service and price are valid. The negotiated service expires automatically at the end of the negotiation interval, and the HRN must periodically re-negotiate (by sending a *Reserve* message) before the expiration to ensure uninterrupted service. Specific services may define different actions on part of the provider regarding the treatment of user packets after the service has expired. Possible actions are: maintaining the current service at the previously negotiated price, maintaining the current service but updating the price unilaterally as required, or transmitting using best effort service. To reduce the signaling overhead, the negotiation interval for a service can be set equal to or a multiple of time periods associated with an underlying protocol, for example, the TCP round-trip time, RSVP [3] refresh time, or RTCP [36] receiver report interval. To reduce control overhead, a minimum negotiation interval should be enforced. Multimedia services should not renegotiate too frequently, to avoid adjusting data rate too often resulting in poor perceived quality.

The negotiation interval affects how a service is priced. A service with a longer negotiation interval may carry a "risk premium" to protect against network dynamics.

Preemption Capability: *Preemption Capability* defines whether the service is pre-emptable or non-pre-emptable. A non-pre-emptable service assures service to the user for the negotiated period. A pre-emptable service is subject to being terminated by the NRN, either asynchronously, or by being allowed to expire at the end of a negotiation interval. For specific services, further refinements may be considered. For example, instead of all the reserved resources being "at risk", resources reserved above a certain base level, or just the cost of reservation may be "at risk". These particulars would be defined by the *SSP* fields.

Reservation Coverage: *Reservation Coverage* indicates the extent of reservation over the flow paths. The reservation can be end-to-end, over contiguous sub-trees where branches may not use or support reservations or for discontiguous segments. In the latter case, referred to as *partial reservations*, reservations may fail on a link, yet the resource reservation request will not be automatically removed for the remaining links.

SSP consists of a list of parameters used to characterize a service, specific to a particular service type. Typical service parameters define the traffic profile the user traffic should adhere to, such as average rate and peak rate, over a certain interval. and the performance promised to the user (average or maximum drop-rate, delay, delay jitter etc.). For some services such as those belonging to diff-serv, the performance requested from a class may be in terms of a qualitative expectation (for example service using EF PHB may be expected to have lower average loss, delay and jitter). In this case, no specific performance parameters are provided.

Price: The *Price* contains a number of sub-fields, including the price being quoted by the service provider for a service, and accumulated charges corresponding to a particular customer Flow Id. More detailed description of *Price* is given in Section 4.

3.2 Protocol Messages

We now describe the RNAP negotiation messages, with some explanation of the sequence in which they are used. The negotiation sequence is represented schematically in Fig. 3.

3.2.1 Query

The HRN uses *Query* messages to request a price quotation from the NRN for one or more services, for each flow or group of flows belonging to the negotiation session. If there is no RNAP session existing between the HRN and the NRN, the HRN generates a random unique *Session Id*. The unique *Session Id* will be used to identify future RNAP messages as belonging to a negotiation session. The HRN will also inform the NRN whether the HRN supports negotiation. The message consists of a set of *Flow Ids*, and one or more Service fields accompanying each *Flow Id*. The HRN specifies a set of requirements with each service, by setting some or all of the *SIP* and *SSP* parameters in the corresponding service identifiers.

3.2.2 Quotation

Upon receiving a *Query* message, the NRN determines the price for each service for which quotations were requested in the *Query* message, and returns a list of *Service* and *Price* pairs inside a *Quotation* message. As stated earlier, we assume the existence of pricing and charging mechanisms here and in the explanation of *Commit* messages, and address the issue in Section 4. A *Query* message with a null *Service* list for one or more *Flow Ids* is interpreted by the NRN as a request for price quotations for all available services, for each such *Flow Id*. The NRN uses default values of *SIP* and *SSP* parameters to determine the price; it does not return quotations for services which have one or more mandatory parameters since the price for these services will depend on the service parameters required and must be provided with a request.

In addition to asynchronously sending *Quotation* messages, as above, the NRN also sends out *Quotation* messages periodically, with a period defined by the *Session Quotation* timer. A *Quotation* sent synchronously message contains price quotations for all services requested.

In general, the NRN sends a *Quotation* message upon receiving a *Query* message, and upon expiry of the *Session Quotation* timer. The timer is reset whenever a *Quotation* message is sent out synchronously, but not when an asynchronous *Quotation* message is sent in response to a *Query*.

If the *Negotiation Capability* flag is false, the HRN and NRN could still exchange an initial pair of *Query* and *Quotation* messages, and negotiate a service with a set of parameters that remains unchanged for the rest of the session. A HRN may re-enable negotiation capability at any time during the session by sending a *Query* or *Reserve* message.

3.2.3 Reserve

The HRN sends a *Reserve* message to apply for services for each flow or group of flows belonging to the negotiation session. A *Reserve* message is sent at the beginning of a session to request services for the first time. Since a service request expires automatically after a *Negotiation Interval* defined for each service, the HRN continues to periodically send *Reserve* messages with a small enough period that none of the requested services expire. Through the *Reserve* message, the HRN applies for services for a particular flow or flow-aggregate, specifying corresponding service identifier identifying the type of service, and a set of *SIP* and *SSP* parameters characterizing the user requirements from that service. In general, each *Reserve* message carries one or more *Flow Id-Service-Price* triples. The function of the *Price* structure in this context is explained in Section 4.1.

When the *Reserve* message includes fewer *Flow Ids* than the previous *Reserve* message, it implies that the *Flow Ids* not included in the new *Reserve* message will be canceled. Similarly, new *Flow Ids* may be added to a *Reserve* message to apply for resources for new flows, and the *Service* field corresponding to a *Flow Id* may be changed to modify the resources requested for an existing flow.

3.2.4 Commit

The *Commit* message is generated by the NRN in response to a *Reserve* message. For each service request specified by a *Flow Id-Service-Price* triple, the NRN determines whether the flows identified by the corresponding *Flow Id* are to be admitted or denied. The admission policy, as stated earlier, is specific to the service, and need not be administered by the NRN. For instance, in a diff-serv service, the bandwidth broker (BB) could make the admission decision, and the NRN simply communicates the admission decision through RNAP. The NRN returns the decision in a list of *Flow Id*, *Service*, *Status* and *Price* 4-tuples.

The *Price* field carries pricing information for the corresponding service. If the flows are admitted, the NRN determines the price for providing the service. If the *Commit* is in response to a re-negotiation *Reserve* request in an ongoing session, the NRN also returns the amount charged for each service in the preceding negotiation period, and the accumulated charge since the beginning of the session.

The *Status* field indicates whether the request for the corresponding service is accepted (rejected, incomplete, or complete). The *Service* identifier is copied in from the *Reserve* message. If the request for that service is rejected, the NRN informs the HRN its reason for denial, by appropriately re-setting parameters in the *Service* identifier. For example, if the service has a service-specific sending rate parameter, and the requested sending rate cannot be supported, the sending rate parameter is set to the maximum sending rate that can be supported. In a way, this informs the sender about the amount of resources available when resources are scarce. The NRN modifies the parameters in a similar manner when the status is *Admit_Incomplete*, to indicate which of the requested parameters has not been granted.

The network could also choose to encourage the HRN to reduce its requirements when network resources are scarce. The requested service from HRN is admitted by setting the *Status* to *Admit_Complete*, but the requested service rate is modified to a smaller value and the price is also set lower than the quoted price as a reward.

3.2.5 Preempt

If a *Service* is set as preemptable (at the benefit of lower price), the NRN may preempt resources allocated previously to this service and make room for the other more important flows. Currently the *Preempt* field in the *Service* identifier is binary, i.e., preemptable or non-preemptable. More preemptation priorities could be supported and allow different flows with different priority levels to be differentiated.

3.2.6 Close

A *Close* message is sent from the HRN to the NRN to tear down the negotiation session between them.

3.2.7 Release

The *Release* message acknowledges the *Close* message and optionally reports to HRN the cumulative charging information for the entire session. This information is for informational purposes, and may not be tied to the actual billing and payment procedures. The NRN releases the resources it had allocated for the session, and sends a *Release* message.

3.3 Sequence of Messages

The messaging sequence for the RNAP-C architecture is shown in Fig. 3. The messaging sequence for RNAP-D is as follows:

1. The HRN sends a *Query* message to the first hop router (FHR). Local and intermediate routers forward the message downstream to the last-hop router (LHR). The LHR determines local service availability and a local price for each service, and initiates a *Quotation* message and sends it upstream. Each intermediate router verifies local availability of each service, and increments the price by the local price that it computes. The FHR returns the *Quotation* message to HRN.

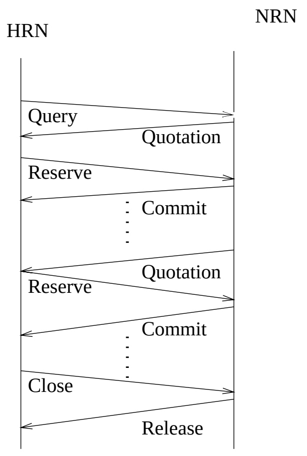


Figure 3: RNAP messaging sequence between HRN and NRN.

As in RNAP-C, *Quotation* messages are also sent periodically to the HRN. The LHR maintains the *Session Quotation* timer, and sends periodic *Quotation* messages hop-by-hop upstream, as above.

2. HRN sends a *Reserve* message to the FHR, and receives a *Commit* message in an identical manner to 2. As the *Commit* message is forwarded upstream, in addition to the committed price being incremented at each router, the incremental charge for each service at that router is added on as well. (Pricing and charging in RNAP-D are considered in more detail in Section 4.2.1.) Subsequently, the HRN periodically re-negotiates resources by sending *Reserve* messages and receiving *Commit* messages in return. *Query* and *Reserve* messages may also be sent asynchronously at any time, as in RNAP-C.

3. To terminate a session, the HRN sends a *Close* message, which is forwarded to the LHR. The LHR sends a *Release* message upstream, and releases *Session Id* and resources. Upstream routers forward the *Release* message towards the HRN and release the *Session Id* and resources.

3.4 Negotiation across Multiple Administrative Domains

In the discussion so far, it has been assumed that the HRN negotiates resources for flows traversing a single domain. We now consider scenarios in which the flows traverse multiple domains. For simplicity, let us first assume that resources are to be negotiated for a session comprising a set of flows between a single source and destination pair. We consider the following scenarios.

3.4.1 End-to-end Resource Negotiation for a Single Customer in RNAP-D

We consider how a customer reserves resources for a flow or group of flows to a particular destination address in RNAP-D. In general, the forwarding of RNAP messages in RNAP-D is similar to RSVP forwarding. The router alert option is turned on in the RNAP message by the originating LRN, so that intermediate LRNs can intercept and process the message before forwarding it to the next-hop LRN. A typical sequence of RNAP messages is as follows.

1. The HRN sends a *Query* message to the first hop LRN (FHL). The FHL forwards the *Query* message downstream to the last-hop LRN (LHL). The LHL determines local service availability and a local price for each service, and initiates a *Quotation* message and sends it upstream. Each intermediate LRN verifies local availability of each service, and increments the price by the local price that it

computes. The FHL returns the *Quotation* message to HRN. Periodic *Quotation* messages are also sent by the LHL hop-by-hop upstream, as above.

2. The HRN periodically sends a *Reserve* message to the FHL, and receives a *Commit* message in an identical manner to the *Query-Quotation* pair. As the *Commit* message is forwarded upstream, in addition to the committed price being incremented at each router, the incremental charge for each service at that router is added on as well. *Query* and *Reserve* messages may also be sent asynchronously at any time in a similar manner.

3.4.2 End-to-end Resource Negotiation for a Single Customer in RNAP-C

We now consider resource reservation for a customer flow traversing multiple network domains; each domain implements RNAP-C, with a controlling NRN. As mentioned earlier, the sequence of messages is identical to that considered earlier for RNAP-D, if each domain is considered to be equivalent to a single node, with the NRN corresponding to the LRN for that node.

The forwarding mechanism in RNAP-C is different from RNAP-D. To forward a RNAP message downstream towards the destination HRN, the NRN first uses the destination host or network address contained in the *Flow Id* to select a neighboring domain border router. The NRN then looks up a local table to map the neighboring domain border router to its associated domain NRN, and forwards the RNAP message to this NRN. The local table is maintained using BGP information. The NRN can be configured as a BGP speaker, or can communicate with other BGP speakers using an interior routing protocol. A neighboring domain NRN address can be sent with BGP UPDATE messages as an optional path attribute by the neighboring domain border routers, allowing each NRN to maintain its local table. The downstream NRN also records the upstream neighboring NRN address as part of the RNAP session state when it receives a RNAP message from it. It uses the upstream NRN address to forward RNAP messages (*Quotation* or *Commit*) in the reverse direction, towards the source HRN.

Similar to RNAP-D, the NRN is responsible for collecting and communicating admission and pricing and charging information for the domain as a whole instead of for a single node (mechanisms for doing this are discussed in Section 3.4.1).

It is also possible that the flow traverses multiple domains some of which implement RNAP-C and others RNAP-D. In this case, the NRN of a RNAP-C domain would talk to the corresponding boundary LRN of an adjoining RNAP-D domain, and the message flow would be as before.

3.5 End-to-end Resource Negotiation with Aggregation of Customer Flows

If end-to-end RNAP reservation is carried out for each customer flow, RNAP agents in the core network may potentially need to process RNAP messages for hundreds of thousands of flows, and maintain state information for each of them. In this section, we first discuss how RNAP messages can be aggregated in the core of the network by allowing RNAP agents to handle reservations for flow-aggregates instead of individual flows. We then address the related issue of how RNAP would be used between two adjoining network domains, to negotiate for resources in bulk for a flow-aggregate.

3.5.1 Aggregation and De-aggregation

We consider the aggregation by an RNAP agent of RNAP messages belonging to a number of different senders on a sink tree, that is senders with the same destination network address (sink tree based aggregation has also been discussed in **??**). The aggregating agent aggregates RNAP messages for user flows which have the same destination network address (obtained from the *Flow Id*)), and also use the same or similar services and have similar negotiation intervals. We consider aggregation first for RNAP-D, and then for RNAP-C.

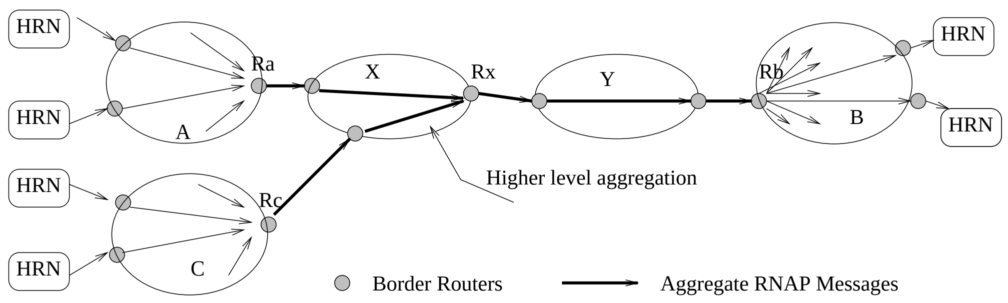


Figure 4: Example RNAP-D message aggregation.

Aggregation and De-aggregation in RNAP-D

Figure 4 illustrates how RNAP message aggregation works in a RNAP-D architecture. Consider the aggregation of *Reserve* messages (this also applies for *Query* messages). At access network A, the border router R_a creates an aggregate *Reserve* message, with the source address set to ‘a’, and the destination network address set to the network address B. It also sets the *Aggregation Flag* to 1 in the *Id* structure, identifying the message as an aggregate message. R_a then forwards the aggregate *Reserve* message hop by hop as in Section 3.4.1. R_a also turns off the router alert option of the incoming per flow messages and tunnels the per-flow *Reserve* messages up to the de-aggregation point, so that per-flow reservation can be resumed in the destination network. In each per-flow *Reserve* message, the address of the aggregator will be included in the *Aggregate Flow Id* field, to enable proper mapping at the de-aggregation point. A per-flow *Reserve* message is encapsulated in an UDP packet with the destination network address set as B, and the port number set to a port reserved for RNAP, and forwarded.

A border router of a domain is a potential deaggregation point for RNAP messages to that domain. Therefore filters are set up at border routers of a domain so as to intercept aggregate RNAP messages as well as tunneled per-flow RNAP messages. For instance, the border router R_b (Fig. 4) of domain B is set up to intercept UDP packets with destination address set to the network address B and port number set to the RNAP port. Once intercepted, aggregate *Reserve* messages and tunneled per-flow messages are sent up to the transport layer. The de-aggregation point will record the mapping between an aggregation flow and per flow messages, by checking the aggregation Flow Id field. The router alert option will be turned on for per-flow *Reserve* messages arriving at R_b, and the messages will be forwarded, allowing per-flow resource reservation within domain B. The aggregate *Reserve* message (identified as such by its *Aggregation Flag*) terminates at the de-aggregation router.

In response, a *Commit* message will be sent upstream for the aggregate *Reserve* message as well each per flow *Reserve* message. The de-aggregation point R_b will decide that the destination address for the per flow *Commit* message is ‘a’, by checking the mapping between the aggregate message and the per flow messages. Each per flow *Commit* message is then encapsulated in a UDP message with destination address ‘a’ and tunneled back to its aggregation point R_a. The aggregate *Commit* message will be forwarded hop by hop by upstream LRN’s until it reaches the aggregation point, and confirms the aggregate *Reserve* request sent by the aggregation agent. There is a similar message flow for RNAP *Quotation* messages in the upstream direction.

The aggregation entity on the source network side is also responsible for de-aggregation of RNAP response messages. That is, it checks the mapping between an aggregate session and per-flow RNAP response messages, and, if it is the origination point for the corresponding aggregate session, it will map the aggregate-level pricing and charging (returned by the aggregate session *Quotation* and *Commit* messages) to the corresponding per-flow prices and charges for individual flows based on the local policy.

Multiple levels of aggregation can occur, so that aggregate messages are aggregated in turn, resulting

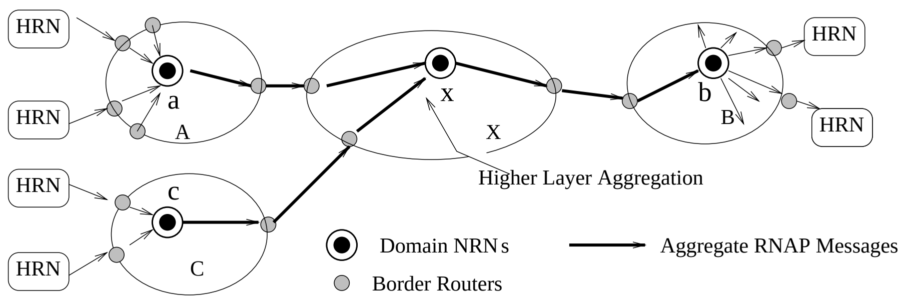


Figure 5: Example RNAP-C message aggregation.

in a progressively thicker aggregate “pipe” towards the root of the sink-tree. For a level two aggregation of several level one RNAP aggregate requests as shown in Fig. 4, node R_x in domain X forms a level two aggregate message with the source address in the *Flow Id* set to ‘x’. Node ’x’ also records the level one requests, and terminates these messages instead of forwarding them. In response, the RNAP agent at the de-aggregation node R_b sends response messages for the level two aggregate towards point ‘x’. At point R_x, the level one response messages are formed by mapping the pricing and charge data from level two aggregate message to individual level one aggregate response massages to send towards R_a and R_c. All the per flow request messages, as before, are tunnelled forward up to node R_b, and per-flow response messages are tunnelled from R_b directly either to R_a or R_c.

Aggregation and De-aggregation in RNAP-C

In the RNAP-C architecture of Fig. 5, the agggregation and de-aggregation entity are NRNs. Once again, we consider the aggregation of *Reserve* messages. At an aggregating NRN ’a’, the aggregate *Reserve* message will be formed and sent domain by domain towards the destination domain NRN ’b’, as in Section 3.4.2. In addition, the destination domain NRN is located through DNS SRV, and the aggregating NRN encapsulates the per flow *Reserve* messages in UDP packet headers and tunnels them directly to the destination domain NRN ’b’.

The destination domain NRN sends a *Commit* messages “hop by hop” (each hop is one domain) upstream towards ’a’ in response to an aggregate *Reserve* message. It will also receive the encapsulated per flow *Reserve* messages from ’a’, process them to perform per-flow reservation, and determine from the *Aggregate Flow Id* field that per-flow response messages are to be encapsulated and tunneled back to ’a’. There is a similar message flow for RNAP *Quotation* messages in the upstream direction. The mapping of pricing and charging information from aggregate session to per flow message is similar to that in RNAP_D. Multiple level agggregation is also supported in RNAP_C.

Aggregation and De-aggregation in Backbone

In the description above, we restrict the aggregation to be performed only for flows that go to the same destination network. In this case, the de-aggregation point can only exist in the destination network, so that per flow processing is avoided inside the core network. We now consider a more general aggregation-deaggregation process, for aggregation of flows or flow-aggregates going to different destination networks.

The advantage of generalizing the aggregation is that a core network domain can aggregate a greater number of flows, and reduce overhead further. The potential disadvantage is that if the de-aggregation point of the aggregated flows is not known in advance, the RNAP agent at each node has to check if any of the component flows diverges from the aggregate at that node, resulting in some per flow state maintaintance and processing in the core network. If the de-aggregation point is known in advance, per flow messages do not need to be processed at each node, but only at the de-aggregation point. However, if the de-aggregation point is inside backbone, the cost may still be prohibitive.

In the discussion of generalized aggregation that follows, we assume that the de-aggregation point(s) of the aggregated sessions is not known by the aggregating agent. However, in order to reduce the processing and state maintainance overhead in the backbone, and avoid per flow message processing, we restrict the first level aggregation (aggregation of individual customer (HRN) flows from an access network, such as A and C in Fig 4) to be performed only for flows to the same destination network, as described previously. Generalized aggregation is allowed only at the second level (aggregation of first level aggregates) and higher.

Generalized aggregation is virtually identical in both RNAP_C and RNAP_D, and we describe it in the context of RNAP_D. Consider a $n+1_{th}$ level aggregation being performed (with n greater than 0) on RNAP *Reserve* messages. The de-aggregation point for a potential new $n+1_{th}$ level aggregation needs to be located first, and the de-aggregator location mechanism can be similar to that proposed in [43]. Initially, the n_{th} RNAP message is sent towards its destination as before, but with the $n+1_{th}$ level aggregating point address inserted into its *Aggregating Flow Id* field by the $n+1_{th}$ level aggregator. A de-aggregation point is defined where the flows forming the $n+1_{th}$ aggregate are diverged (are determined to have different next-hop nodes). A router decides itself to be a $n+1_{th}$ level de-agregator if it finds that different aggregate flows with the same *Aggregating Flow Id* are destinated to the different next hop routers. A service-denying *Commit* message will be triggered to be sent towards the aggregating point with the denial reason given as a split point located. This allows the aggregation agent to learn the address of the de-aggregation point. The n_{th} level RNAP *reserve* message will continue to be sent downstream of the potential split point and processed as described before. The denying *Commit* message due to a new split point location will not affect the reservation that has been made by the n_{th} RNAP *reserve* message. Different n_{th} level RNAP *reserve* may carry back different split point addresses. The $n+1_{th}$ level aggregator chooses the split point with the shortest distance (how to find it?) to itself as the de-aggregator for the $n+1_{th}$ level aggregation.

Subsequently, the $n+1_{th}$ level aggregating point creates the $n+1_{th}$ level aggregate messages based on the n_{th} level aggregate messages and the destination of the $n+1_{th}$ level aggregate message is set to the de-aggregator address. It also re-sets the *Aggregate Flag* of the n_{th} level messages to 0, sets *Aggregate Flow Id* in these messages to its own address. It will tunnel the n_{th} level request messages directly to the de-aggregation point by turning off the rotuer alert option, and encapsulating the n_{th} level aggregate messages to the de-aggregator. The de-aggregation entity terminates the $n+1th$ level request messages, and re-activates the n_{th} messages (by setting the Aggregate Flag to 1) and forwards them. It also records and maintains the mapping of $n+1_{th}$ level flow messages to the n_{th} level flow messages.

The de-aggregation entity serves as the aggregation point for n_{th} level response messages (*Quoation* and *Commit*). Since it knows the address of the $n+1_{th}$ level aggregation agent (from received $n+1_{th}$ level messages), it tunnels these response messages directly to the aggregation point, using UDP encapsulation as described previously.

If a route change occurs due to a topology update, RNAP agents at the affected routers will send a service-denying *Commit* message for each aggregate RNAP session to the aggregation point, with the denial reason given as route change. The aggregator will then send n_{th} level RNAP messages hop-by-hop (unencapsulated) until a new de-aggregation point to be detected.

The aggregation in RNAP_C is similar to RNAP_D, and the n_{th} flow messages will be tunneled to the de-aggregation point directly once it is detected.

The maximum number of state entries a de-aggregation point needs to maintain is roughly proportional to the number of neighboring domains for the aggregating domain, which is normally a small number. So processing and state maintanance at the de-aggregation point is tractable. The RNAP processing in the core network, which may include resource allocation, admission control, classification, policing, as well as scheduling, is proportional to the number of nodes times the number of tree aggregate flows, a much higher number. Hence RNAP protocol overhead can be further reduced in the backbone due to the higher level aggregation of flows not destinated to the same desination network.

Overhead Reduction due to Aggregation

As a result of the aggregation of RNAP messages, the message processing overhead and the storage of the RNAP state information are greatly reduced in the core network. Since per flow messages need to be tunneled to the destination network, so the RNAP message transmission bandwidth is not reduced, and actually slightly increased because of the extra aggregation messages. But since RNAP messages are updated with a relatively long interval, this is not a major concern compared with the bandwidtht hat will be consumed by the data flows.

3.5.2 Resource Negotiation for Flow-Aggregates and Advanced Reservation

At the aggregation point, the NRN acts as the client negotiator (or HRN) for an aggregate session in negotiations with the downstream NRN (for example, referring to Fig. 5 again, the aggregating NRN 'a' negotiates with NRN 'x'). In general, the client negotiator will negotiate resources for an aggregate session corresponding to the per-flow reservation requests. To avoid frequent re-negotiation, however, it is likely that the client negotiator will increment or decrement the requested resources with some minimum granularity. When the sum of per-flow requests approaches the resources reserved (or reach some threshold) for the aggregate, the client negotiator will reserve an additional block of resources. Similarly, the requested reservation is decremented in blocks as required. The larger the block, the less frequently the aggregate session needs to be re-negotiated, but a higher holding cost is incurred for resources which may be under-utilized. In general, negotiation of resources in blocks results in a fairly static service, and periodical re-negotiations, if any, would occur with a much longer negotiation interval. Hence, price *Quotation* messages for the aggregate session will probably only be sent asynchronously in response to *Query* messages, when an additional block of resources needs to be reserved or removed.

The NRN at an aggregation point may also forecast a certain demand to a particular destination network, and could negotiate a large block of resources in advance, using the advance reservation mechanism. The HRN or client NRN indicates an advance reservation using the *Starting Time* and *Ending Time* fields in the Service description. The server NRN initializes session state at the conclusion of the advance negotiations, and maintains the state until the actual transmission has been completed.

If the client negotiator chooses to cancel part or all of a reservation made in advance, it can re-negotiate with the server negotiator to try to 'sell back' previously reserved resources at an agreed price. The price eventually agreed upon would probably reflect any cancellation or holding cost fee the server negotiator wishes to charge. The server negotiator may also offer to buy back resources reserved in advance, for more important usage.

3.6 Negotiation for a Multicast Session

RNAP request messages for a multicast session will negotiate services for flows to multiple destinations. In this case, the *Flow Id* field carries the IP multicast address. On receiving such a message, the NRN (or LRN) communicates with a multicast routing protocol to determine if the multicast tree diverges into two or more branches within the NRN-administered domain (or at the LRN-router). If so, multiple copies of the message are formed and forwarded to different directions.

The NRN or LRN maintains session state allowing it to aggregate RNAP response messages traveling upstream at divergence points on the multicast tree. As response messages are aggregated, the pricing and charging information from individual response messages are summed to obtain the corresponding information for the aggregate message.

4 Pricing and Charging

The main RNAP messages, *Query*, *Reserve*, *Quotation* and *Commit*, all contain a common *Price* structure, used to convey pricing and charging information. We first briefly discuss the purpose of the various *Price* fields, and how they are used in RNAP messaging. We then consider the issues of formulating end-to-end prices and charging customer flows accordingly. We address these issues within both the RNAP-D and RNAP-C architectures, and also discuss pricing and charging across multiple network domains. We also briefly consider the scenario in which sender and receiver HRNs share the charges for services used, and consider charging in a multicast session.

We end this section with a proposal for a specific strategy for pricing a BBE service at a single network point. This lies outside the scope of the RNAP protocol and architecture, but taken together with the global pricing and charging mechanisms, it would constitute a complete and viable pricing system.

4.1 Pricing Structure and its Use in RNAP Messages

The *Price* structure carried by RNAP messages consists of the following fields: *New Price*, *Current Charge*, *Accumulated Charge*, and *HRN Data*. There is a *Price* structure corresponding to each *Service* structure carried in a RNAP message. The *New Price* field contains the price quoted by the network provider to the negotiating HRN for the next negotiation period. The units of the quoted price are service-specific. A reasonable unit could be "currency/Mb", so that the charge is computed according to the volume of the data transmitted. Alternatively, the unit could be "currency/time", so that the charge is computed according to the time of usage at a specific data transmission rate. In this case, the HRN can expect to be charged an amount equal to the *New Price* multiplied by the length of the negotiation period.

The *Current Charge* field contains the amount charged by the network provider for the preceding negotiation period. This field should have an unit of currency, for example, dollars, but the specific unit is service specific. The *Accumulated Charge* field contains the total amount charged by the network provider since the beginning of the negotiation session. The accumulated charge is carried to protect against the loss of *Commit* messages.

The *HRN Data* fields in a message pertain to the HRN from which the message originates (usually the negotiating HRN, but we will shortly discuss a situation in which the partner or peer HRN originates the message). The *HRN Account* field identifies the account to which charges are to be debited. The corresponding *Charging Fraction* field indicates the fraction of the total charge to be borne by the HRN. If for example, the negotiating HRN wishes to be responsible for half of the charges, (in the understanding that the peer HRN will be responsible for the other half), it sets the *Charging Fraction* to 0.5. We return to this issue in more detail in Section 4.3. The minimum and maximum data rate fields are included to allow the sender and receiver HRNs to reach a basic agreement about the desired transmission rate. With respect to the sender HRN, the data rates represent the minimum and maximum sending rates the sender is willing and able to transmit. With respect to the receiver HRN, these rates indicate the minimum and maximum data rates the receiver is willing and able to receive. The *other options* field is intended to carry other information that could be used to facilitate negotiation.

In general, the *Price* structure accompanies a corresponding *Service* structure in protocol messages. *Query* and *Quotation* messages carry a set of *Service-Price* pairs corresponding to all the services for which price quotations are requested, and *Reserve* and *Commit* messages each carry a set of *Flow Id-Service-Price* triples corresponding to the services being provided for the flows or flow aggregates belonging to the negotiation session. Pricing and charging information follows the following basic flow: after a session has been opened, the negotiating HRN sends a *Query* message carrying *Service-Price* pairs. The HRN indicates how much of the charge for each service it is willing to bear by setting the *HRN Data.HRN Charge Fraction* field accordingly, and may also indicate its budget for a particular service by setting the *New Price* field. If the *Query* message has a null *Service* list, the HRN may still wish to indicate how much of the total charge it is willing to bear by including a single *Price* structure by itself in the *Query* message, with the *HRN Charge Fraction* set. The network responds to the *Query* message with a *Quotation* message in which the *New Price* fields are set to the price quoted for each service, if it is possible to determine it on the basis of the received *Query*. The HRN then requests one or more services through a *Reserve* message. As with the *Query* message, it can use the *Price* structures in the *Reserve* message to indicate the fraction of the charge it is willing to bear for each service. The network responds with a *Commit* message, committing or denying the requests, and setting the *Price:New Price* field for each *Flow Id-Service* pair to the committed price.

Subsequently, the network sends periodic *Quotation* messages to quote the updated price for available services, and the HRN and network re-negotiate services by exchanging *Reserve* and *Commit* messages. The *Price* structures in these messages are used as before. In addition, the *Price* :*Current Charge* field in the *Commit* message is used to carry the charges for the corresponding *Flow Id-Service* pairs in the preceding negotiation interval.

4.2 Price and Charge Formulation

In the previous section, we discussed how price and charge information are communicated to the HRN through RNAP messages. We now consider the issue of arriving at the contracted price to be quoted for a flow receiving a particular service in a given negotiation period, and computing the charge for the service at the end of the period. Let us first define the data structure to be used by the network to maintain the price and charge information. We call this the *Session Charge State*:

```
Session Charge State  =  Session Id
                      :  Flow Charge State 1
                      :  Flow Charge State 2
                      :  .
                      :  .
                      :  Flow Charge State n

Flow Charge State  =  Flow Id
                   :  New Price
                   :  Current Charge
                   :  Accumulated Charge
```

In general, prices are re-calculated periodically, based on network traffic characteristics, and this period is independent of the RNAP negotiation interval. The *New Price* structure maintains the current price structure to be applied for the service received by a particular flow. The *New Price* structure may consist of several fields in order to reflect a complex pricing strategy such as that presented in Section 4.5, and is hence more complicated than the single *New Price* field carried in RNAP messages, which simply quote the estimated price to the HRN. The *New Price* fields remain unchanged during a negotiation interval, and are updated at the end of a negotiation period if prices have changed at some time during the interval. At the end of each negotiation period, the *Current Charge* field is re-computed using the *New Price* structure for that period. The *Accumulated Charge* holds the accumulated charge since the beginning of the session, and is incremented by the *Current Charge* at the end of a negotiation period.

The *Session Charge State* information is maintained by different entities, and used in different ways, depending on the RNAP architecture. We consider the centralized and distributed architectures separately.

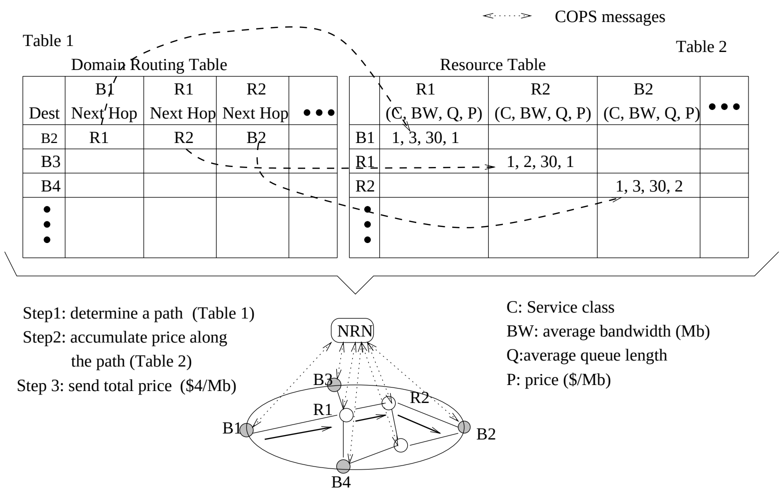

COPS messages
Table 1
Domain Routing Table
Table 2
Resource Table
Step1: determine a path (Table 1)
Step2: accumulate price along the path (Table 2)
Step 3: send total price ($4/Mb)
NRN
B1
B2
B3
B4
R1
R2
C: Service class
BW: average bandwidth (Mb)
Q:average queue length
P: price ($/Mb)

Table 1 — Domain Routing Table

Dest	B1 Next Hop	R1 Next Hop	R2 Next Hop	• • •
B2	R1	R2	B2	
B3				
B4				
• • •				

Table 2 — Resource Table

	R1 (C, BW, Q, P)	R2 (C, BW, Q, P)	B2 (C, BW, Q, P)	• • •
B1	1, 3, 30, 1			
R1		1, 2, 30, 1		
R2			1, 3, 30, 2	
• • •				

Figure 6: Price formulation in RNAP-C

4.2.1 Price and Charge Formulation in RNAP-D

In the RNAP-D (distributed) architecture, each router-LRN maintains charging state information for the flows passing through it, based on prices computed at the router. At the beginning of a negotiation period (and also in response to a *Query* message), the last hop LRN originates a *Quotation message*. The *Quotation* message is sent hop-by-hop back towards the first-hop LRN. At each LRN, the *Price:New Price* fields in the message are incremented according to the current *New Price* computed for the corresponding service at the LRN. In Section 4.5, we discuss a specific local pricing strategy in which a set of prices is computed for each service. In this case, some mapping behavior may have to be defined to obtain a single increment for the quoted *New Price*. When the *Quotation* message arrives at the negotiating HRN, it carries the total quoted price for each service.

Similarly, *Commit* messages originate at the last-hop LRN, and are sent hop-by-hop back to the first-hop LRN. In this case, the *New Price*, *Current Charge*, and *Accumulated Charge* fields are all incremented at each router-LRN on the way.

4.2.2 Price Formulation in RNAP-C

When the centralized negotiation architecture is used, the local charging state information for a domain is maintained by the NRN. The price formulation strategy is a much more open-ended problem. Various alternatives may be considered, and different domains may apply different local policies. The NRN may compute a price based on the service specifications alone. The price could be fixed, or modified based on the time of day, etc. In general, if the price charged to a flow needs to depend on the network state and the flow path, we consider the following three approaches:

1. The NRN makes the admission decision and decides the price for a service, based on the network topology, routing and configuration policies, and network load. In this case, the NRN sits at a router that belongs to a link-state routing domain (for example an OSPF area) and has an identical link state database as other routers in the domain. This allows it to calculate all the routing tables of all other routers in the domain using Dijkstra's algorithm. A similar idea has been explored in [41] in a different context.

   The NRN maintains a domain routing table which finds any flow route that either ends in its own domain, or uses its domain as a transmit domain (Fig. 6). The domain routing table will be updated whenever the link state database is changed. A NRN also maintains a resource table, which allows it to keep track of the availability and dynamic usage of the resources (bandwidth, buffer space). In general, the resource table stores resource information for each service provided at a router. The resource table allows the NRN to compute a local price at each router (for instance, using the usage-based pricing strategy described in Section 4.5). For a particular service request, the NRN first looks up the path on which resources are requested using the domain routing table, and then uses the per-router prices to compute the accumulated price along this path. The resource table also facilitates monitoring and provisioning of resources at the routers. To enable the NRN to collect resource information, routers in the domain periodically report local state information (for instance, average buffer occupancy and bandwidth utilization) to the NRN. A protocol such as COPS [30] can be used for this purpose.

To compute the charge for a flow, ingress routers maintain per-flow (or aggregated flow from neighboring domains) state information about the data volume transmitted during a negotiation period. This information is periodically transmitted to the NRN, allowing the NRN to compute the charge for the period. The NRN uses the computed price and charge to maintain charging state information for each RNAP session.

2. Prices are computed at the network boundary, and communicated to the NRN. For price calculation, there are two alternatives.

   One alternative is that the ingress router periodically computes a price for each service class and ingress-egress pair. The calculation is based on service specifications and local per-service demand at the ingress router; internal router states along the flow path are not taken into account.

   The other alternative allows internal router load to be taken into account. Probe messages are sent periodically from an egress router to all ingress routers. A probe message carries per-service *Price* structures which accumulate prices hop-by-hop at each router in a similar manner to Section 4.2.1.

   In both of the above cases, the ingress router maintains per-flow state information that includes the per-flow price (the price charged to the service class the flow belongs to), as well as the per-flow data volume entering the domain. This information is transmitted every negotiation period to the NRN, which computes the charge and is responsible for the messaging.

3. Price formulation takes place through a intra-domain signaling protocol. If resource reservation for a particular service in a domain is performed through a dynamic resource reservation protocol, such as RSVP or YESSIR[4], the price information is collected through the periodic messages of the reservation protocol, and stored at the ingress router. For example, the RSVP PATH message and RTCP [36] messages in YESSIR can collect pricing information. If the ingress router is responsible for sending the price information to the NRN, the price accumulated from a domain will be send back to ingress router along with the RSVP RESV message. Such an implementation, utilizing RSVP, is described in 5. Communication between the ingress router and NRN occurs as discussed in the first scenario.

In the above schemes, we assume that a domain has one NRN. A domain could also have multiple NRNs, each NRN residing at an ingress router. In this case, the ingress router does not need to send periodic per-session reports to a centralized NRN, and pricing, charging, and RNAP messaging are done directly from the ingress router. Reliability concerns make a more distributed architecture (multiple NRNs, or RNAP-D) preferable. But some management goals (for instance, all NRNs in one domain need to have coherent view of the resource at internal routers to allow them to make correct admission decisions) may make a centralized policy more attractive.

4.2.3 Charge Formulation for Multiple Domains and Flow Aggregates

When a customer flow spans more than one administrative domain, each domain computes incremental prices and charges for the flow using its own pricing strategy and architecture, and the total end-to-end price and charge are obtained in a hop-by-hop manner (with each domain representing a single hop) as in Section 4.2.1.

When a set of flows enter a domain as a flow-aggregate, the NRN (or network domain as a whole in RNAP-D) carries out messaging and charging as if the aggregate belonged to a single customer. The NRN in the aggregating domain (or LRN at the aggregation point) is responsible for mapping the total charge into charges for individual customer flows or flow-aggregates.

4.3 Shared Charging

Let us assume that the sender HRN negotiates services, but the receiver pays part of the bill. We consider end-to-end services across multiple domains, and assume for convenience the centralized architecture in each domain - the equivalent situation in a distributed architecture can be understood by replacing the NRN with a router.

The sender HRN sets the *Price:HRN Data.HRN Charge Fraction* fields in the service identifier in *Query* and *Reserve* messages according to the fraction of total charges it is willing to bear. Any *Query* or *Reserve* message with *Price:HRN Data.HRN Charge Fraction* less than 1 is forwarded automatically by the last hop NRN to the receiver. The receiver HRN copies the *Query* or *Reserve* message into a modified *Query* or *Reserve* message and indicates its willingness to pay by setting the *Price:HRN Data.HRN Charge Fraction* field to (1-negotiating *HRN Charge Fraction*). It may indicate its unwillingness to be responsible for the entire amount by setting *Price:HRN Data.HRN Charge Fraction* to a smaller value. It could also agree to bear the entire charge, but indicate an upper limit on the price it is willing to pay by setting the *Price:New Price field*. [A service is established only if the total williness to pay from the sender and receiver is greater than one.].

The receiver HRN sends its modified *Query* or *Reserve* message to the last hop NRN. A modified *Query* message is read by the last hop NRN to generate a *Quotation* message which is forwarded by intermediate NRNs back to the sender, where it serves as feedback to the sender HRN about the willingness to pay of the receiver. A modified *Reserve* message is similarly read by the last hop NRN and used to generate a *Commit* message either accepting or denying the service requested by the *Reserve* message. If the respective *Price:HRN Data.HRN Charge Fraction* fields in the *Query* (or *Reserve*) messages received from the sender and receiver HRNs add up to less than 1, the service request is denied, and the *Status:Reasons* field is set accordingly. The *Commit* message is forwarded upstream through intermediate NRNs, updating *Status* and *Price* fields along the way.

In receiver negotiation with sender bearing part of the charges, a similar sequence of messages is used, except that the flow of information is in the reverse direction.

If receiver participates in negotiation, other than indicating its willingness to pay, the receiver could also set the *Price:HRN Data.Maximum Data Rate* and *Price:HRN Data.Minimum Data Rate* fields to convey to the sender the minimum sending rate it requires and the maximum rate that it can handle. This allows the receiver, for example, to indicate to the sender that it cannot handle a rate offered by the sender, and in general, provide the sender guidelines for the negotiation process.

4.4 Multicast Charging

In a multicast session, either sender or the receivers could negotiate separately, or they could both participate in negotiation.

If the sender is solely responsible for negotiation and payment, the messaging sequence is similar to the simple scenario considered for unicast. The sender HRN determines a service request based on price quotations from the NRN, and on feedback about received quality from the receivers. A similar messaging sequence is also followed when the receiver negotiates and is responsible for payments. The receiver's *Reserve* message is based on its knowledge of sender traffic formats. In both the above case, the sender and receiver can learn about each other's capabilities and requirements by end-to-end *Query* and *Quotation* messages contained in the *HRN Data* fields.

In receiver negotiation with partial or full sender payment, the receiver HRN learns about the sender's willingness to pay through end-to-end *Query* and *Quotation* messages, as in Section 4.3. The sender may specify a maximum expenditure through the *Price:New Price*. An example of this kind of negotiation may be when a company multicasts a commercial advertisement. The receiver adjusts its received transmission according to the sender budget and network conditions.

If the sender negotiates, with partial or full receiver payment, the sender receives feedback about each receiver's willingness to pay, as discussed in Section 4.3. The sender negotiates resources based on the overall demand willingness to pay, and may request partial reservation on some paths on which receivers have a low willingness to pay.

4.5 Pricing Strategy

In the previous sections, we discussed the mechanisms for pricing in RNAP, including the pricing structure used by RNAP messages, and the formulation of end-to-end prices and charges in the RNAP architecture. We assumed the existence of specific pricing strategies or rules for the negotiated service. As discussed earlier, specific pricing strategies are outside the scope of the RNAP protocol itself. However, for completeness, we briefly describe a pricing strategy that could work with the RNAP protocol presented in more detail in [10].

The pricing strategy used in the network is based on the competitive market model [24]. The competitive market model defines two kinds of agents: consumers and producers. Consumers seek resources from producers, and producers create or own the resources. The exchange rate of a resource is called its price. Prices are set where the amount of resource demanded equals the amount of resources supplied. The routers are considered as the producers and own the link bandwidth and buffer space for each output port. The flows (individual flows or aggregate of flows) are considered as consumers who consume resources. Prices are computed locally at routers, and collated to form an end-to-end price using the RNAP protocol.

The price computation is performed periodically, with a price updating interval τ, and the price within each interval is kept constant to provide some degree of predictability to users. The router has multiple output ports and supports multiple levels of service. A price is computed separately for the buffer space and link bandwidth associated with each output port. We also assume that the router is partitioned to provide separate link bandwidth and buffer space for each class of service. The total demand for link bandwidth is based on the aggregate bandwidth reserved on the link for a price computation interval, and the total demand for the buffer space at an output port is the average buffer occupancy during the interval. The supply bandwidth and buffer space need not be equal to the installed capacity; instead, they are the targeted bandwidth and buffer space utilization.

We decompose the total charge computed at a router into three components: *holding charge*, *usage charge*, and *congestion charge*. The usage charge is determined by the actual resources consumed, the level of service guaranteed to the user. The usage price is set such that it allows a retail network to recover the cost of the purchase from the wholesale market, and various static costs associated with the service.

The holding charge is imposed if some resources (buffer space or bandwidth) are set aside for a user, even when the user traffic does not utilize the resources. It reflects the potential revenue lost by the provider because it can only sell the allotted resources at the usage charge of a lower service level (e.g., scheduling packets from lower level service classes).

A congestion charge is imposed if congestion is deduced, that is, the demand (in terms of buffer space or bandwidth) exceeds supply (the targeted buffer space or bandwidth).

The detailed form of each of the component charges is presented in [10]. Based on the above price formulation strategy, a router arrives at a price structure for a particular session (flow or flow-aggregate), at the end of each price updating interval. The total charge for a session can be represented as:

$$session\_charge = \sum_{n=1}^{N}(holding\_charge(n) + usage\_charge(n) + congestion\_charge(n))$$

where n represents a negotiation period during a session, and N is total number of intervals spanned by a session.

5 Implementation

5.1 Overview

In this section, we describe an implementation of the RNAP protocol and architecture in a test-bed network. Our purpose was to provide a preliminary demonstration of the protocol, and in the future work we will implement the RNAP functionality completely and in detail. For simplicity, the distributed (RNAP_D) architecture was assumed, and the RSVP signaling protocol was extended to provide the important RNAP mechanisms of periodic re-negotiation, price quotation, and charging. An RNAP agent (LRN) was implemented at each node. Two types of service were implemented : the traditional best-effort service, and the Controlled Load (CL) service proposed within the int-serv model. RSVP is a receiver-driven protocol, and accordingly, the HRN at the receiver side acts as the resource negotiator.

Although our implementation was highly simplified, it allowed us to demonstrate several features: the periodic RNAP negotiation process including resource negotiation and pricing and charging; the stability of the usage-sensitive pricing algorithm and its effectiveness in controlling congestion; the adaptation of user applications in response to changes in network conditions and hence in the service price.

5.2 Protocol Implementation

The RNAP *Quotation*, *Reserve* and *Commit* information are embedded in RSVP *Path*, *Resv* and *ResvErr* messages. The implementation does not incorporate the RNAP *Query* message at present; this is not critical, particularly since only one service type is being offered to the user. The functionality of the *Quotation* and *Commit* messages is somewhat different from the functionality described earlier. Since *Commit* messages cannot easily be sent periodically in this implementation framework, and since RSVP reservation is based on flows, the *Quotation* message carries periodic charging information (in the *Price:Current Charge* and *Price:Accumulated Charge* fields) instead of the *Commit* message. Currently the RNAP negotiation period is set to be the same as the RSVP refresh period. The default refresh interval is 30 seconds.

The sequence of messages is as follows:

1. RSVP *Path* messages, with embedded RNAP *Quotation* information are sent periodically from the sender-LRN towards the receiver-LRN. At present, the *Quotation* message only contains the *Price* structure, with quoted price and accumulated charge information. In general, it would contain various *SSP* and *SIP* fields. As a *Path* message passes each node, the *Price* field is updated to add the price computed at the local node and the incremental charge for the previous period.

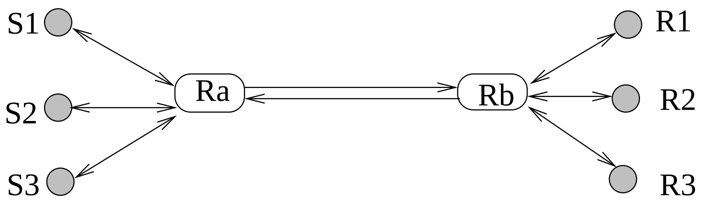


Figure 7: Testbed setup

2. The HRN at the receiver receives the *Path* message and sends a RSVP *Resv* request, with embedded RNAP *Reservation* information. The *Price* received from *Path* is copied into the *Price* field that will be sent with the RSVP *Resv* message to sender direction, with the *Price:HRN Data* field updated to indicate receiver information.

3. When a RSVP *Resv* request is rejected, an RSVP *ResvErr* message is sent to the receiver HRN, with embedded *Commit* information. This information includes "bandwidth available" information in the *Price:HRN Data.Maximum Rate* field.

In the present implementation, user traffic is served either best-effort, or using the Controlled Load service model. Resource reservation on each hop is performed using Class-Based Queueing (CBQ) [34]. CBQ partitions and shares link bandwidth using hierarchically structured classes. Each class has its own queue and is assigned a share of the link bandwidth. A child class can borrow bandwidth from its parent class as long as excess bandwidth is available. Weighted-round robin (WRR) scheduling is used to serve packets from classes with the same priority.

One or more flows may request CL service at the router. When the corresponding RNAP session is established at the router, a corresponding new CBQ class is created under the parent class that is configured for CL service. All CBQ classes for CL service are served with same high priority, using WRR scheduling.

A *Policy Element*, called the *Price Element*, is defined to hold *Price* structure 4. As with other *Policy Elements*, the *Price Element* is opaque to RSVP and is only understood by policy peers. The *Price Element* is embedded within the *POLICY_DATA* objects [32, 31] of *Path* messages, *Resv* messages and *ResvErr* messages.

Each node has a resident RNAP agent, which was implemented in our experiment with the Local Policy Decision Point (LPDP) proposed in the COPS architecture [30, 31]. The RNAP agent periodically computes a set of prices (for the single CL service class) based on traffic through the node. The RNAP agent also maintains the *Session Charge State* for each session, and updates it whenever a *Path* message passes through the node.

Since per flow queuing is used, we do not enforce congestion charging when per-flow queues individually overflow. Users are simply penalized by dropping packets. However, the total link usage relative to the total link bandwidth is monitored, and congestion charge is levied when necessary.

5.3 Experimental Setup

In order to demonstrate some important functionality of the RNAP protocol at a router, as well as to evaluate the effectiveness of the pricing algorithm presented in Section 4.5, the above implementation was tested over a very simple topology, consisting of 2 nodes connected by a 10 Mb/s link, schematically represented in Fig. 7. The `rsvpd` version 4 from ISI [33] was extended to support RNAP, as discussed above. The ALTQ package [35] has been used for scheduling and queue management. In particular, CBQ is used together with `rsvpd`, and CBQ states are monitored for pricing and charge purpose.

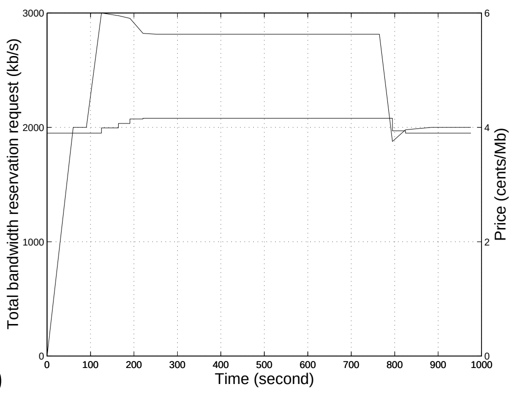
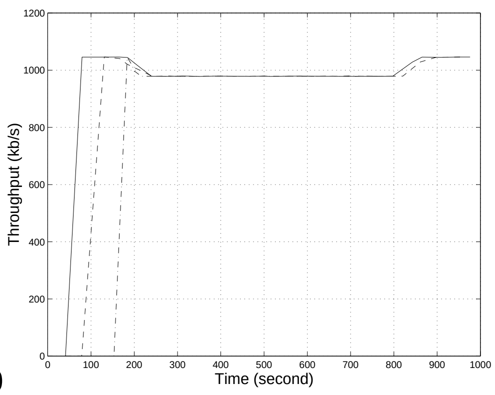
a)  b) 

Figure 8: a) The system price and total reservation requests and b) the throughput of each session shown as a function of the time

Three RSVP sessions were established end to end, and shared the same output interface of the link. To create different levels of network load, a simple source model was used in each session to continuously send UDP packets. The packet generation rate was tunable to allow a session to adapt to any data rate it intended to send. At any sending rate, the packets were generated periodically. Background traffic was sent using best effort service.

Out of the interface capacity of 10 Mb/s of each interface of Ra and Rb, 4 Mb/s was configured to support the high priority CL service, and the remaining bandwidth was configured as default class and used for best effort service. To look closely at the congestion control, pricing and charging functions of RNAP, the CBQ states were monitored at node Ra.

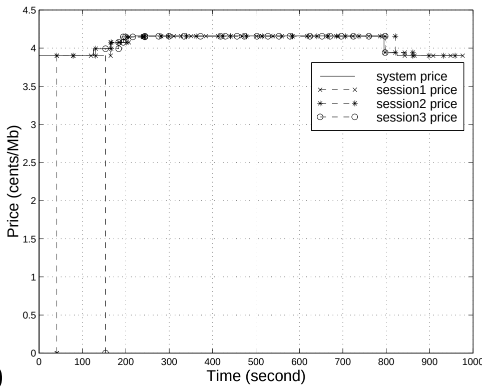
During the experiment, each of the three application HRNs individually tried to optimize its own utility. Since our main purpose was to test the performance of RNAP at a router, all the HRNs were given the same budget, and each HRN asked for the maximum bandwidth it could afford during each negotiation period. Price formulation at a node was in terms of the holding, usage and congestion prices, as discussed previously. The HRN was quoted the current total price, as representing an upper bound on the charge for the ensuing negotiation period in the absence of congestion. The targeted utilization of the link was set at 70% (2.8 Mb/s); if the total demand on the link exceeded this threshold, congestion pricing was enforced as described previously.

We assumed a service roughly as expensive (per unit bandwidth) as a telephone cable. Assuming a charge of 10 c/min, and a capacity of 64 kb/s, the usage price is set as 2.6 c/Mb. Assuming that the next lower level of service is charged at 5 c/min, or 1.3 c/Mb, the holding price is set at 1.3 c/Mb (can be any price proportional to this). The congestion price will not be updated when the difference between demand and supply is within 5% of the supply.

We assume that the budget available to each application is such that it can request a sending rate of 1Mb/s at the initial quoted price. The application will reduce the requested bandwidth when the price increases. The performance metrics considered are: the price dynamics and its influence on the total QoS requirement from applications, the charge (network revenue) and the throughput of a flow during each negotiation period.

5.4 Analysis of Results

Fig. 8a shows the total price charged at node Ra at different network loads, and the total bandwidth request in response to changes in price. When the total bandwidth request is less than the supply bandwidth, 2.8 Mb/s, the price is set at the minimum level of 3.9 c/Mb ($p_h = 1.3$ c/Mb, $p_u = 2.6$ c/Mb). At around $t = 130$ seconds, the total reservation exceeds the supply bandwidth and the congestion price is enforced for bandwidth reservation. After three negotiation periods, the total reservation recovers to close to the

a) 

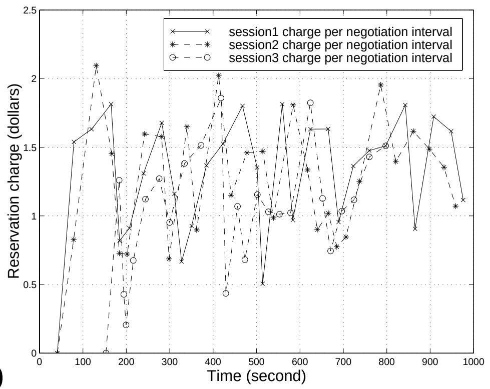
b) 

Figure 9: a) The price for each session and b) the session charge (system revenue) for each negotiation interval shown as a function of the time

supply level, and the new, stabilized price is about 20% greater than the minimum price. At $t = 700$ seconds, one of the sessions is terminated, and the total reservation is now much smaller than the supply. The price starts to reduce and after two negotiation periods the price stabilizes at 3.9 c/Mb again.

Fig. 8b shows the change in per-session throughput with time. It is seen that all the sessions share bandwidth fairly if all the sessions have the same traffic format, price sensitivity and budget. If some session can afford a higher price, it could reserve more bandwidth and gain higher throughput. Since CBQ is allowed to borrow bandwidth from the parent class, the throughput of each session is slightly higher than the requested reservation, around 978 kb/s against 938 kb/s at time $t = 860$ seconds. We see that the stabilized reservation rate is only 6.2% lower than the maximum requested rate.

Fig. 9a shows the computed price at the beginning of each period. The price for each period is set as the system price at the beginning of the period, and will be kept constant during a negotiation interval.

Fig. 9b shows the charge for each negotiation period. Since in RSVP new intervals are initiated randomly from 15 s to 45 s, the charge for each period varies randomly. Since all the sessions have equal budgets and similar initial sending rates, they all have approximately the same charge within a similar length of negotiation period.

6 Related Work

In this section we briefly discuss related research work in three main areas: resource reservation and allocation mechanisms; adaptive applications; billing and pricing in the network.

6.1 Resource Reservation and Allocation

Current research in providing QoS support in the Internet is mainly based on two architectures defined by IETF: Per-flow based *integrated services* (int-serv) [5], and class-based *differentiated service* (diff-serv) [7]. In both architectures, implementations should include a mechanism by which the user can request specific network services, and thus acquire network resources. Per-flow resource reservation in int-serv is generally implemented through the RSVP reservation protocol [3]. Implementation of resource reservation for diff-serv is a subject of ongoing research, and various approaches have been proposed [11]. In general, RSVP and the implementations of diff-serv lack integrated mechanisms by which the user can select one out of a spectrum of services, and re-negotiate resource reservations dynamically. They also do not integrate the pricing and billing mechanisms which must accompany such services.

6.2 Adaptive Applications

There has been a lot of recent research on adaptation of the sending rates of multimedia applications in response to available network resources [9], which relies on signaling mechanisms such as packet loss rates for feedback. The orientation of these methods is different from ours, since they assume no QoS support and no usage sensitive pricing of network services. The frequent and passive rate adjustment can severely degrade the multimedia quality, and sometimes an application is even not able to maintain its minimum QoS requirement.

6.3 Pricing and Billing in the Network

Microeconomic principles has been applied to various network traffic management problems. The studies in [13][17][19][20][27] are based on a maximization process to determine the optimal resource allocation such that the utility (a function that maps a resource amount to a satisfaction level) of a group of users is maximized. These approaches normally rely on a centralized optimization process, which does not scale. Also, some of the algorithms assume some knowledge of the user's utility curves and truthful revelation by users of their utility curves, which may not be practical.

In [12][16][21][22][25], the resources are priced to reflect demand and supply. The pricing model in these approaches is usage-sensitive - it has been shown that usage-sensitive pricing results in higher utilization than traditional flat rate pricing [12]. Some of these methods are limited by their reliance on a well-defined statistical model of source traffic, and are generally not intended to adapt to changing traffic demands.

The scheme presented in [22] is more similar to our work in that it takes into account the network dynamics (session join or leave) and source traffic characteristics (VBR). It also allows different equilibrium price over a different time period, depending on the different user resource demand. However, congestion is only considered during admission control. Our pricing algorithm has two congestion-dependent components, namely congestion due to excessive resource reservation (holding cost) and congestion due to network usage (usage cost).

In general, the work cited above differs from ours in that it does not enter into detail about the negotiation process and the network architecture, and mechanisms for collecting and communicating locally computed prices. Our work is more concerned with developing a flexible and general framework for resource negotiation and pricing and billing, decoupled from specific network service protocols and pricing and resource allocation algorithms. Our work can therefore be regarded as complementary with some of the cited work.

Karsten et al [26] introduce a charging and payment scheme for RSVP-based QoS reservations. A significant difference from our work is the absence of an explicit price quotation mechanism - instead, the user accepts or rejects the estimated charge for a reservation request. Also, the scheme is coupled to a particular service environment (int-serv), whereas our goal is to develop a more flexible negotiation protocol usable with different service models.

7 Summary and Future Work

The overall objective of this paper has been to develop a protocol and architecture which enables network service negotiation for multiple delivery services and environments. The RNAP protocol enables service negotiation between user applications and the access network, as well as between adjoining network domains. The protocol permits negotiation and communication of QoS specifications, user traffic profiles, admission of service requests, and pricing and charging information for requested services. The periodic (as well as asynchronous) re-negotiation framework of the protocol enables dynamic, usage sensitive adaptation of service parameters and pricing by the network, if required, and also enables the user application to respond to changes in application requirements. At the same time, the framework provides sufficient flexibility to support users with limited negotiation capability, or with a requirement for very static and predictable service specifications.

A pair of alternate protocol architectures has been described. The RNAP-D architecture is based on a distributed, per-node model, while the RNAP-C architecture concentrates the negotiation functionality at a centralized entity, the NRN. The first architecture is tailored to delivery services with relatively strict flow control and "hard", or quantitative QoS specifications. The second architecture may be better suited for delivery service models dealing with service negotiations with a coarser granularity (multiple flows or flow-aggregates) and providing statistical or qualitative specifications. In either case, the architecture is scalable because it does not assume service reservation with a particular granularity, and incorporates mechanisms for flow aggregation. The two architectures use the same set of RNAP messages, and can co-exist and inter-operate across multiple administrative domains.

The protocol and architectures provide mechanisms for local or incremental price computation at a single point in the network, collation of local prices in order to compute end-to-end prices along different routes, and communication of prices and charges to the client. Several price and charge collation mechanisms have been described for the distributed and centralized architectures, and end-to-end pricing and charging across several administrative domains has also been discussed. An algorithm for local pricing at a router has been discussed in detail, but the pricing and charging mechanisms in the protocol are independent of the specific pricing algorithm used. A protype implementation of the important RNAP functionalities has been described, along with some preliminary measurement results.

The important directions for future development of this work include a more detailed study of mechanisms associated with resource aggregation, development of more sophisticated pricing strategies, deployment of the full functionality of RNAP in a large scale network, and more extensive tests for the performance of RNAP signalling.

References

[1] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," RFC 2212, Sept. 1997.

[2] J. Wroclawski, "Specification of the controlled load quality of service," RFC 2211, Sept. 1997.

[3] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol (RSVP) - version 1 functional specification," RFC 2205, Sept. 1997.

[4] P. Pan and H. Schulzrinne, "YESSIR: A simple reservation mechanism for the Internet", In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98)*, Cambridge, England, July 1998.

[5] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," Request for Comments (Informational) 1633, Internet Engineering Task Force, June 1994.

[6] K. Nichols, V. Jacobson, and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet," Internet Draft, Internet Engineering Task Force, Nov., 1997

[7] K. Nichols and S. Blake, "Differentiated services operational model and definitions," Internet Draft, Internet Engineering Task Force, Feb. 1998. Work in progress.

[8] Y. Bernet, R. Yavatkar, P. Ford, F. Baker, L. Zhang, K. Nichols, M. Speer, and B. Braden, "Interoperation of RSVP/Int-Serv and diff-serv networks," Internet Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

[9] X. Wang, H. Schulzrinne, "Comparison of adaptive Internet multimedia applications," in *IEICE Transactions on Communications*, June, 1999.

[10] X. Wang and H. Schulzrinne, “Incentive-Compatible Adaptation of Internet Real-Time Mult imedia,” Technical Report, Columbia University, Dec. 1999.

[11] Internet 2 Bandwidth Broker Information, http://www.merit.edu/working.groups/i2-qbone-bb.

[12] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, “Pricing in computer networks: Motivation, formulation, and example,” *IEEE/ACM Transactions on Networking*, vol. 1, pp 614-27, Dec. 1993.

[13] J. F. MacKie-Mason and H. Varian, “Pricing Congestible Network Resources,” *IEEE J. Select. Areas Commun.*,, vol. 13, no. 7, pp 1141-9, Sept. 1995.

[14] J. F. MacKie-Mason, L. Murphy, and J. Murphy, “The role of responsive pricing in the Internet,” *Internet Economics*, J. Bailey and L. McKnight eds., MIT Press, 1997, pp 279-303.

[15] A. Gupta, D. O. Stal, and A. B. Whinston, “A priority pricing approach to manage multiple-service class networks in real-Time,” *Internet Economics*, J. Bailey and L. McKnight eds., MIT Press, 1997, pp 323-52.

[16] N. Anerousis and A. A. Lazar, “A framework for pricing virtual circuit and virtual path services in atm networks”, *ITC-15*, pp. 791 - 802, 1997.

[17] A. Hafid, G. V. Bochmann and B. Kerherve,“A quality of service negotiation procedure for distributed multimedia presentational applications,” In *Proceedings of the Fifth IEEE International Symposium On High Performance Distributed Computing (HPDC-5)*, Syracuse, New York, 1996.

[18] T. F. Abdelzaher, E. M. Atkins, and K. Shin, “QoS negotiation in real-time systems and its application to automated flight control,” To appear in *IEEE Transactions on Software Engineering*, 1999.

[19] H. Jiang and S. Jordan, “A pricing model for high speed networks with guaranteed quality of service,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Fransisco, California), Mar. 1996.

[20] S. Low and P. Varaiya, “An algorithm for optimal service provisioning using resource pricing,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Toronto, Canada), June 1994.

[21] D. F. Ferguson, C. Nikolaou, and Y. Yemini, “An economy for flow control in computer networks,” in Proceedings of the *Conference on Computer Communications (IEEE Infocom)*, (Ottawa, Canada), pp. 110-118, IEEE, Apr. 1989.

[22] E. W. Fulp, D. S. Reeves, “Distributed network flow control based on dynamic competive markets,”In *Proceedings International Conference on Network Protocol (ICNP’98)*, Austin Texas, Oct. 13-16, 1998.

[23] D. Reininger, D. Raychaudhuri and M. Ott, “Market based bandwidth allocation policies for QoS control in broadband networks,” In *Proceedings of the First International Conference on Information and Computation Economies (ICE-98)*, pages 101-110, Qct., 1998.

[24] H. Varian, “Microeconomic Analysis,” Third Edition, 1993. W.W. Norton & pany.

[25] J. Sairamesh, “Economic paradigms for information systems and networks”, PhD thesis, Columbia University, New York 1997.

[26] M. Karsten, J. Schmitt, L. Wolf, and R. Steinmetz, “An embedded charging approach for RSVP,” *The Sixth International Workshop on Quality of Service (IWQoS’98)*, pp 91-100, Napa, California, USA.

[27] F. P. Kelly, A.K. Maulloo and D.K.H. Tan, “Rate control in communication networks: shadow prices, proportional fairness and stability,” *Journal of the Operational Research Society* 49 (1998), 237-252.

[28] C. Lee, J. Lehoczky, R. Rajkumar and D. Siewiorek, “On Quality of Service Optimization with Discrete QoS Options,” *Proceedings of the IEEE Real-time Technology and Applications Symposium*, June 1999.

[29] G. Bianchi, A.T. Campbell, and R.R.-F. Liao, “On utility-fair adaptive services in wireless networks, ” *6th International Workshop on Quality of Service (IEEE/IFIP IWQOS’98)* , Napa Valley, CA, May 1998.

[30] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry, “The COPS (Common Open Policy Service) Protocol,” Internet Draft, Internet Engineering Task Force, Feb. 24, 1999. Work in progress.

[31] R. Yavatkar, D. Pendarakis, and R. Guerin, “A framework for policy-based admission control,” Internet Draft, Internet Engineering Task Force, Nov. 1998. Work in progress.

[32] S. Herzog, “RSVP extensions for policy control,” Internet Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

[33] RSVP software release, ftp://ftp.isi.edu/rsvp/release.

[34] Floyd, S., and Jacobson, V., “Link-sharing and Resource Management Models for Packet Networks”, *IEEE/ACM Transactions on Networking*, Vol. 3, No. 4, pp. 365-386, August 1995.

[35] K. Cho, ALTQ: Alternate Queueing for FreeBSD.

[36] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: a transport protocol for real-time applications,” RFC 1889, Jan. 1996.

[37] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, “Service location protocol,” Request for Comments (Proposed Standard) 2165, Internet Engineering Task Force, June 1997.

[38] A. Gulbrandsen, P. Vixie, “A DNS RR for specifying the location of services (DNS SRV),” RFC 2052 , Oct. 1996.

[39] F. Baker, B. Lindell, and M. Talwar, “RSVP cryptographic authentication,” Internet Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

[40] R. Atkinson and S. Kent, “Security architecture for the Internet protocol,” Request for Comments (Proposed Standard) 2401, Internet Engineering Task Force, Nov. 1998.

[41] O. Schelen, S. Pink, “Resource reservation agents in the Internet,” in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp 153-156, July, 1998.

[42] P. Pan, E. Hahne, H. Schulzrinne, “BGRP: A Tree-Based Aggregation Protocol for Inter-Domain Reservations”. submitted.

[43] F. Baker, et al, “ Aggregation of RSVP for IPv4 and IPv6 Reservations”, Internet Engineering Task Force, June 1999.

[44] RADB, A Distributed Database for Internet Routing Registry. http://www.radb.net/docs/.

[45] R. Vaccaro, “Digital control, a state space approach”, McGraw Hill, New York, 1995