                Specification of Guaranteed Quality of Service

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   This memo describes the network element behavior required to deliver
   a guaranteed service (guaranteed delay and bandwidth) in the
   Internet. Guaranteed service provides firm (mathematically provable)
   bounds on end-to-end datagram queueing delays. This service makes it
   possible to provide a service that guarantees both delay and
   bandwidth. This specification follows the service specification
   template described in [1].

Introduction

   This document defines the requirements for network elements that
   support guaranteed service. This memo is one of a series of
   documents that specify the network element behavior required to
   support various qualities of service in IP internetworks. Services
   described in these documents are useful both in the global Internet
   and private IP networks.

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119.

   This document is based on the service specification template given in
   [1]. Please refer to that document for definitions and additional
   information about the specification of qualities of service within
   the IP protocol family.

In brief, the concept behind this memo is that a flow is described
using a token bucket and given this description of a flow, a service
element (a router, a subnet, etc) computes various parameters
describing how the service element will handle the flow's data. By
combining the parameters from the various service elements in a path,
it is possible to compute the maximum delay a piece of data will
experience when transmitted via that path.

It is important to note three characteristics of this memo and the
service it specifies:

   1. While the requirements a setup mechanism must follow to achieve
   a guaranteed reservation are carefully specified, neither the
   setup mechanism itself nor the method for identifying flows is
   specified. One can create a guaranteed reservation using a
   protocol like RSVP, manual configuration of relevant routers or a
   network management protocol like SNMP. This specification is
   intentionally independent of setup mechanism.

   2. To achieve a bounded delay requires that every service element
   in the path supports guaranteed service or adequately mimics
   guaranteed service. However this requirement does not imply that
   guaranteed service must be deployed throughout the Internet to be
   useful. Guaranteed service can have clear benefits even when
   partially deployed. If fully deployed in an intranet, that
   intranet can support guaranteed service internally. And an ISP
   can put guaranteed service in its backbone and provide guaranteed
   service between customers (or between POPs).

   3. Because service elements produce a delay bound as a result
   rather than take a delay bound as an input to be achieved, it is
   sometimes assumed that applications cannot control the delay. In
   reality, guaranteed service gives applications considerable
   control over their delay.

   In brief, delay has two parts: a fixed delay (transmission delays,
   etc) and a queueing delay. The fixed delay is a property of the
   chosen path, which is determined not by guaranteed service but by
   the setup mechanism. Only queueing delay is determined by
   guaranteed service. And (as the equations later in this memo
   show) the queueing delay is primarily a function of two
   parameters: the token bucket (in particular, the bucket size b)

and the data rate (R) the application requests. These two values
are completely under the application's control. In other words,
an application can usually accurately estimate, a priori, what
queueing delay guaranteed service will likely promise.
Furthermore, if the delay is larger than expected, the application
can modify its token bucket and data rate in predictable ways to
achieve a lower delay.

End-to-End Behavior

   The end-to-end behavior provided by a series of network elements that
   conform to this document is an assured level of bandwidth that, when
   used by a policed flow, produces a delay-bounded service with no
   queueing loss for all conforming datagrams (assuming no failure of
   network components or changes in routing during the life of the
   flow).

   The end-to-end behavior conforms to the fluid model (described under
   Network Element Data Handling below) in that the delivered queueing
   delays do not exceed the fluid delays by more than the specified
   error bounds. More precisely, the end-to-end delay bound is $[(b-M)/R*(p-R)/(p-r)]+(M+Ctot)/R+Dtot$ for $p>R>=r$, and $(M+Ctot)/R+Dtot$ for
   $r<=p<=R$, (where b, r, p, M, R, Ctot, and Dtot are defined later in
   this document).

      NOTE: While the per-hop error terms needed to compute the end-to-
      end delays are exported by the service module (see Exported
      Information below), the mechanisms needed to collect per-hop
      bounds and make the end-to-end quantities Ctot and Dtot known to
      the applications are not described in this specification. These
      functions are provided by reservation setup protocols, routing
      protocols or other network management functions and are outside
      the scope of this document.

   The maximum end-to-end queueing delay (as characterized by Ctot and
   Dtot) and bandwidth (characterized by R) provided along a path will
   be stable. That is, they will not change as long as the end-to-end
   path does not change.

   Guaranteed service does not control the minimal or average delay of
   datagrams, merely the maximal queueing delay. Furthermore, to
   compute the maximum delay a datagram will experience, the latency of
   the path MUST be determined and added to the guaranteed queueing
   delay. (However, as noted below, a conservative bound of the latency
   can be computed by observing the delay experienced by any one
   packet).

   This service is subject to admission control.

Motivation

   Guaranteed service guarantees that datagrams will arrive within the
   guaranteed delivery time and will not be discarded due to queue
   overflows, provided the flow's traffic stays within its specified
   traffic parameters. This service is intended for applications which
   need a firm guarantee that a datagram will arrive no later than a
   certain time after it was transmitted by its source. For example,
   some audio and video "play-back" applications are intolerant of any
   datagram arriving after their play-back time. Applications that have
   hard real-time requirements will also require guaranteed service.

   This service does not attempt to minimize the jitter (the difference
   between the minimal and maximal datagram delays); it merely controls
   the maximal queueing delay. Because the guaranteed delay bound is a
   firm one, the delay has to be set large enough to cover extremely
   rare cases of long queueing delays. Several studies have shown that
   the actual delay for the vast majority of datagrams can be far lower
   than the guaranteed delay. Therefore, authors of playback
   applications should note that datagrams will often arrive far earlier
   than the delivery deadline and will have to be buffered at the
   receiving system until it is time for the application to process
   them.

   This service represents one extreme end of delay control for
   networks. Most other services providing delay control provide much
   weaker assurances about the resulting delays. In order to provide
   this high level of assurance, guaranteed service is typically only
   useful if provided by every network element along the path (i.e. by
   both routers and the links that interconnect the routers). Moreover,
   as described in the Exported Information section, effective provision
   and use of the service requires that the set-up protocol or other
   mechanism used to request service provides service characterizations
   to intermediate routers and to the endpoints.

Network Element Data Handling Requirements

   The network element MUST ensure that the service approximates the
   "fluid model" of service. The fluid model at service rate R is
   essentially the service that would be provided by a dedicated wire of
   bandwidth R between the source and receiver. Thus, in the fluid
   model of service at a fixed rate R, the flow's service is completely
   independent of that of any other flow.

The flow's level of service is characterized at each network element
by a bandwidth (or service rate) R and a buffer size B. R represents
the share of the link's bandwidth the flow is entitled to and B
represents the buffer space in the network element that the flow may
consume. The network element MUST ensure that its service matches
the fluid model at that same rate to within a sharp error bound.

The definition of guaranteed service relies on the result that the
fluid delay of a flow obeying a token bucket (r,b) and being served
by a line with bandwidth R is bounded by b/R as long as R is no less
than r. Guaranteed service with a service rate R, where now R is a
share of bandwidth rather than the bandwidth of a dedicated line,
approximates this behavior.

Consequently, the network element MUST ensure that the queueing delay
of any datagram be less than $b/R+C/R+D$, where C and D describe the
maximal local deviation away from the fluid model. It is important
to emphasize that C and D are maximums. So, for instance, if an
implementation has occasional gaps in service (perhaps due to
processing routing updates), D needs to be large enough to account
for the time a datagram may lose during the gap in service. (C and D
are described in more detail in the section on Exported Information).

    NOTE: Strictly speaking, this memo requires only that the service
    a flow receives is never worse than it would receive under this
    approximation of the fluid model. It is perfectly acceptable to
    give better service. For instance, if a flow is currently not
    using its share, R, algorithms such as Weighted Fair Queueing that
    temporarily give other flows the unused bandwidth, are perfectly
    acceptable (indeed, are encouraged).

Links are not permitted to fragment datagrams as part of guaranteed
service. Datagrams larger than the MTU of the link MUST be policed
as nonconformant which means that they will be policed according to
the rules described in the Policing section below.

Invocation Information

Guaranteed service is invoked by specifying the traffic (TSpec) and
the desired service (RSpec) to the network element. A service
request for an existing flow that has a new TSpec and/or RSpec SHOULD
be treated as a new invocation, in the sense that admission control
SHOULD be reapplied to the flow. Flows that reduce their TSpec
and/or their RSpec (i.e., their new TSpec/RSpec is strictly smaller
than the old TSpec/RSpec according to the ordering rules described in
the section on Ordering below) SHOULD never be denied service.

The TSpec takes the form of a token bucket plus a peak rate (p), a minimum policed unit (m), and a maximum datagram size (M).

The token bucket has a bucket depth, b, and a bucket rate, r. Both b and r MUST be positive. The rate, r, is measured in bytes of IP datagrams per second, and can range from 1 byte per second to as large as 40 terabytes per second (or close to what is believed to be the maximum theoretical bandwidth of a single strand of fiber). Clearly, particularly for large bandwidths, only the first few digits are significant and so the use of floating point representations, accurate to at least 0.1% is encouraged.

The bucket depth, b, is also measured in bytes and can range from 1 byte to 250 gigabytes. Again, floating point representations accurate to at least 0.1% are encouraged.

The range of values is intentionally large to allow for the future bandwidths. The range is not intended to imply that a network element has to support the entire range.

The peak rate, p, is measured in bytes of IP datagrams per second and has the same range and suggested representation as the bucket rate. The peak rate is the maximum rate at which the source and any reshaping points (reshaping points are defined below) may inject bursts of traffic into the network. More precisely, it is a requirement that for all time periods the amount of data sent cannot exceed M+pT where M is the maximum datagram size and T is the length of the time period. Furthermore, p MUST be greater than or equal to the token bucket rate, r. If the peak rate is unknown or unspecified, then p MUST be set to infinity.

The minimum policed unit, m, is an integer measured in bytes. All IP datagrams less than size m will be counted, when policed and tested for conformance to the TSpec, as being of size m. The maximum datagram size, M, is the biggest datagram that will conform to the traffic specification; it is also measured in bytes. The flow MUST be rejected if the requested maximum datagram size is larger than the MTU of the link. Both m and M MUST be positive, and m MUST be less than or equal to M.

   The guaranteed service uses the general TOKEN_BUCKET_TSPEC
   parameter defined in Reference [8] to describe a data flow's
   traffic characteristics. The description above is of that
   parameter. The TOKEN_BUCKET_TSPEC is general parameter number
   127. Use of this parameter for the guaranteed service TSpec
   simplifies the use of guaranteed Service in a multi-service
   environment.

The RSpec is a rate R and a slack term S, where R MUST be greater
than or equal to r and S MUST be nonnegative. The rate R is again
measured in bytes of IP datagrams per second and has the same range
and suggested representation as the bucket and the peak rates. The
slack term S is in microseconds. The RSpec rate can be bigger than
the TSpec rate because higher rates will reduce queueing delay. The
slack term signifies the difference between the desired delay and the
delay obtained by using a reservation level R. This slack term can
be utilized by the network element to reduce its resource reservation
for this flow. When a network element chooses to utilize some of the
slack in the RSpec, it MUST follow specific rules in updating the R
and S fields of the RSpec; these rules are specified in the Ordering
and Merging section. If at the time of service invocation no slack
is specified, the slack term, S, is set to zero. No buffer
specification is included in the RSpec because the network element is
expected to derive the required buffer space to ensure no queueing
loss from the token bucket and peak rate in the TSpec, the reserved
rate and slack in the RSpec, the exported information received at the
network element, i.e., Ctot and Dtot or Csum and Dsum, combined with
internal information about how the element manages its traffic.

The TSpec can be represented by three floating point numbers in
single-precision IEEE floating point format followed by two 32-bit
integers in network byte order. The first floating point value is
the rate (r), the second floating point value is the bucket size (b),
the third floating point is the peak rate (p), the first integer is
the minimum policed unit (m), and the second integer is the maximum
datagram size (M).

The RSpec rate term, R, can also be represented using single-
precision IEEE floating point.

The Slack term, S, can be represented as a 32-bit integer. Its value
can range from 0 to (2**32)-1 microseconds.

When r, b, p, and R terms are represented as IEEE floating point
values, the sign bit MUST be zero (all values MUST be non-negative).
Exponents less than 127 (i.e., 0) are prohibited. Exponents greater
than 162 (i.e., positive 35) are discouraged, except for specifying a
peak rate of infinity. Infinity is represented with an exponent of
all ones (255) and a sign bit and mantissa of all zeroes.

Exported Information

   Each guaranteed service module MUST export at least the following
   information. All of the parameters described below are
   characterization parameters.

A network element's implementation of guaranteed service is
characterized by two error terms, C and D, which represent how the
element's implementation of the guaranteed service deviates from the
fluid model. These two parameters have an additive composition rule.

The error term C is the rate-dependent error term. It represents the
delay a datagram in the flow might experience due to the rate
parameters of the flow. An example of such an error term is the need
to account for the time taken serializing a datagram broken up into
ATM cells, with the cells sent at a frequency of $1/r$.

   NOTE: It is important to observe that when computing the delay
   bound, parameter C is divided by the reservation rate R. This
   division is done because, as with the example of serializing the
   datagram, the effect of the C term is a function of the
   transmission rate. Implementors should take care to confirm that
   their C values, when divided by various rates, give appropriate
   results. Delay values that are not dependent on the rate SHOULD
   be incorporated into the value for the D parameter.

The error term D is the rate-independent, per-element error term and
represents the worst case non-rate-based transit time variation
through the service element. It is generally determined or set at
boot or configuration time. An example of D is a slotted network, in
which guaranteed flows are assigned particular slots in a cycle of
slots. Some part of the per-flow delay may be determined by which
slots in the cycle are allocated to the flow. In this case, D would
measure the maximum amount of time a flow's data, once ready to be
sent, might have to wait for a slot. (Observe that this value can be
computed before slots are assigned and thus can be advertised. For
instance, imagine there are 100 slots. In the worst case, a flow
might get all of its N slots clustered together, such that if a
packet was made ready to send just after the cluster ended, the
packet might have to wait 100-N slot times before transmitting. In
this case one can easily approximate this delay by setting D to 100
slot times).

If the composition function is applied along the entire path to
compute the end-to-end sums of C and D (Ctot and Dtot) and the
resulting values are then provided to the end nodes (by presumably
the setup protocol), the end nodes can compute the maximal datagram
queueing delays. Moreover, if the partial sums (Csum and Dsum) from
the most recent reshaping point (reshaping points are defined below)
downstream towards receivers are handed to each network element then
these network elements can compute the buffer allocations necessary

to achieve no datagram loss, as detailed in the section Guidelines
for Implementors. The proper use and provision of this service
requires that the quantities Ctot and Dtot, and the quantities Csum
and Dsum be computed. Therefore, we assume that usage of guaranteed
service will be primarily in contexts where these quantities are made
available to end nodes and network elements.

The error term C is measured in units of bytes. An individual
element can advertise a C value between 1 and 2**28 (a little over
250 megabytes) and the total added over all elements can range as
high as (2**32)-1. Should the sum of the different elements delay
exceed (2**32)-1, the end-to-end error term MUST be set to (2**32)-1.

The error term D is measured in units of one microsecond. An
individual element can advertise a delay value between 1 and 2**28
(somewhat over two minutes) and the total delay added over all
elements can range as high as (2**32)-1. Should the sum of the
different elements delay exceed (2**32)-1, the end-to-end delay MUST
be set to (2**32)-1.

The guaranteed service is service_name 2.

The RSpec parameter is numbered 130.

Error characterization parameters C and D are numbered 131 and 132.
The end-to-end composed values for C and D (Ctot and Dtot) are
numbered 133 and 134. The since-last-reshaping point composed values
for C and D (Csum and Dsum) are numbered 135 and 136.

Policing

   There are two forms of policing in guaranteed service. One form is
   simple policing (hereafter just called policing to be consistent with
   other documents), in which arriving traffic is compared against a
   TSpec. The other form is reshaping, where an attempt is made to
   restore (possibly distorted) traffic's shape to conform to the TSpec,
   and the fact that traffic is in violation of the TSpec is discovered
   because the reshaping fails (the reshaping buffer overflows).

   Policing is done at the edge of the network. Reshaping is done at
   all heterogeneous source branch points and at all source merge
   points. A heterogeneous source branch point is a spot where the
   multicast distribution tree from a source branches to multiple
   distinct paths, and the TSpec's of the reservations on the various
   outgoing links are not all the same. Reshaping need only be done if
   the TSpec on the outgoing link is "less than" (in the sense described
   in the Ordering section) the TSpec reserved on the immediately
   upstream link. A source merge point is where the distribution paths

or trees from two different sources (sharing the same reservation)
merge. It is the responsibility of the invoker of the service (a
setup protocol, local configuration tool, or similar mechanism) to
identify points where policing is required. Reshaping may be done at
other points as well as those described above. Policing MUST not be
done except at the edge of the network.

The token bucket and peak rate parameters require that traffic MUST
obey the rule that over all time periods, the amount of data sent
cannot exceed M+min[pT, rT+b-M], where r and b are the token bucket
parameters, M is the maximum datagram size, and T is the length of
the time period (note that when p is infinite this reduces to the
standard token bucket requirement). For the purposes of this
accounting, links MUST count datagrams which are smaller than the
minimum policing unit to be of size m. Datagrams which arrive at an
element and cause a violation of the the M+min[pT, rT+b-M] bound are
considered non-conformant.

At the edge of the network, traffic is policed to ensure it conforms
to the token bucket. Non-conforming datagrams SHOULD be treated as
best-effort datagrams. [If and when a marking ability becomes
available, these non-conformant datagrams SHOULD be ''marked'' as
being non-compliant and then treated as best effort datagrams at all
subsequent routers.]

Best effort service is defined as the default service a network
element would give to a datagram that is not part of a flow and was
sent between the flow's source and destination. Among other
implications, this definition means that if a flow's datagram is
changed to a best effort datagram, all flow control (e.g., RED [2])
that is normally applied to best effort datagrams is applied to that
datagram too.

   NOTE: There may be situations outside the scope of this document,
   such as when a service module's implementation of guaranteed
   service is being used to implement traffic sharing rather than a
   quality of service, where the desired action is to discard non-
   conforming datagrams. To allow for such uses, implementors SHOULD
   ensure that the action to be taken for non-conforming datagrams is
   configurable.

Inside the network, policing does not produce the desired results,
because queueing effects will occasionally cause a flow's traffic
that entered the network as conformant to be no longer conformant at
some downstream network element. Therefore, inside the network,
network elements that wish to police traffic MUST do so by reshaping
traffic to the token bucket. Reshaping entails delaying datagrams
until they are within conformance of the TSpec.

Reshaping is done by combining a buffer with a token bucket and peak
rate regulator and buffering data until it can be sent in conformance
with the token bucket and peak rate parameters. (The token bucket
regulator MUST start with its token bucket full of tokens). Under
guaranteed service, the amount of buffering required to reshape any
conforming traffic back to its original token bucket shape is
b+Csum+(Dsum*r), where Csum and Dsum are the sums of the parameters C
and D between the last reshaping point and the current reshaping
point. Note that the knowledge of the peak rate at the reshapers can
be used to reduce these buffer requirements (see the section on
"Guidelines for Implementors" below). A network element MUST provide
the necessary buffers to ensure that conforming traffic is not lost
at the reshaper.

   NOTE: Observe that a router that is not reshaping can still
   identify non-conforming datagrams (and discard them or schedule
   them at lower priority) by observing when queued traffic for the
   flow exceeds b+Csum+(Dsum*r).

If a datagram arrives to discover the reshaping buffer is full, then
the datagram is non-conforming. Observe this means that a reshaper
is effectively policing too. As with a policer, the reshaper SHOULD
relegate non-conforming datagrams to best effort. [If marking is
available, the non-conforming datagrams SHOULD be marked]

   NOTE: As with policers, it SHOULD be possible to configure how
   reshapers handle non-conforming datagrams.

Note that while the large buffer makes it appear that reshapers add
considerable delay, this is not the case. Given a valid TSpec that
accurately describes the traffic, reshaping will cause little extra
actual delay at the reshaping point (and will not affect the delay
bound at all). Furthermore, in the normal case, reshaping will not
cause the loss of any data.

However, (typically at merge or branch points), it may happen that
the TSpec is smaller than the actual traffic. If this happens,
reshaping will cause a large queue to develop at the reshaping point,
which both causes substantial additional delays and forces some
datagrams to be treated as non-conforming. This scenario makes an
unpleasant denial of service attack possible, in which a receiver who
is successfully receiving a flow's traffic via best effort service is
pre-empted by a new receiver who requests a reservation for the flow,
but with an inadequate TSpec and RSpec. The flow's traffic will now
be policed and possibly reshaped. If the policing function was
chosen to discard datagrams, the best-effort receiver would stop
receiving traffic. For this reason, in the normal case, policers are
simply to treat non-conforming datagrams as best effort (and marking

them if marking is implemented). While this protects against denial
of service, it is still true that the bad TSpec may cause queueing
delays to increase.

   NOTE: To minimize problems of reordering datagrams, reshaping
   points may wish to forward a best-effort datagram from the front
   of the reshaping queue when a new datagram arrives and the
   reshaping buffer is full.

   Readers should also observe that reclassifying datagrams as best
   effort (as opposed to dropping the datagrams) also makes support
   for elastic flows easier. They can reserve a modest token bucket
   and when their traffic exceeds the token bucket, the excess
   traffic will be sent best effort.

A related issue is that at all network elements, datagrams bigger
than the MTU of the network element MUST be considered non-conformant
and SHOULD be classified as best effort (and will then either be
fragmented or dropped according to the element's handling of best
effort traffic). [Again, if marking is available, these reclassified
datagrams SHOULD be marked.]

Ordering and Merging

   TSpec's are ordered according to the following rules.

   TSpec A is a substitute ("as good or better than") for TSpec B if (1)
   both the token rate r and bucket depth b for TSpec A are greater than
   or equal to those of TSpec B; (2) the peak rate p is at least as
   large in TSpec A as it is in TSpec B; (3) the minimum policed unit m
   is at least as small for TSpec A as it is for TSpec B; and (4) the
   maximum datagram size M is at least as large for TSpec A as it is for
   TSpec B.

   TSpec A is "less than or equal" to TSpec B if (1) both the token rate
   r and bucket depth b for TSpec A are less than or equal to those of
   TSpec B; (2) the peak rate p in TSpec A is at least as small as the
   peak rate in TSpec B; (3) the minimum policed unit m is at least as
   large for TSpec A as it is for TSpec B; and (4) the maximum datagram
   size M is at least as small for TSpec A as it is for TSpec B.

   A merged TSpec may be calculated over a set of TSpecs by taking (1)
   the largest token bucket rate, (2) the largest bucket size, (3) the
   largest peak rate, (4) the smallest minimum policed unit, and (5) the
   smallest maximum datagram size across all members of the set. This
   use of the word "merging" is similar to that in the RSVP protocol
   [10]; a merged TSpec is one which is adequate to describe the traffic
   from any one of constituent TSpecs.

A summed TSpec may be calculated over a set of TSpecs by computing
(1) the sum of the token bucket rates, (2) the sum of the bucket
sizes, (3) the sum of the peak rates, (4) the smallest minimum
policed unit, and (5) the maximum datagram size parameter.

A least common TSpec is one that is sufficient to describe the
traffic of any one in a set of traffic flows. A least common TSpec
may be calculated over a set of TSpecs by computing: (1) the largest
token bucket rate, (2) the largest bucket size, (3) the largest peak
rate, (4) the smallest minimum policed unit, and (5) the largest
maximum datagram size across all members of the set.

The minimum of two TSpecs differs according to whether the TSpecs can
be ordered. If one TSpec is less than the other TSpec, the smaller
TSpec is the minimum. Otherwise, the minimum TSpec of two TSpecs is
determined by comparing the respective values in the two TSpecs and
choosing (1) the smaller token bucket rate, (2) the larger token
bucket size (3) the smaller peak rate, (4) the smaller minimum
policed unit, and (5) the smaller maximum datagram size.

The RSpec's are merged in a similar manner as the TSpecs, i.e. a set
of RSpecs is merged onto a single RSpec by taking the largest rate R,
and the smallest slack S. More precisely, RSpec A is a substitute
for RSpec B if the value of reserved service rate, R, in RSpec A is
greater than or equal to the value in RSpec B, and the value of the
slack, S, in RSpec A is smaller than or equal to that in RSpec B.

Each network element receives a service request of the form (TSpec,
RSpec), where the RSpec is of the form (Rin, Sin). The network
element processes this request and performs one of two actions:

 a. it accepts the request and returns a new Rspec of the form
    (Rout, Sout);
 b. it rejects the request.

The processing rules for generating the new RSpec are governed by the
delay constraint:

$$Sout + b/Rout + Ctoti/Rout <= Sin + b/Rin + Ctoti/Rin,$$

where Ctoti is the cumulative sum of the error terms, C, for all the
network elements that are upstream of and including the current
element, i. In other words, this element consumes (Sin - Sout) of
slack and can use it to reduce its reservation level, provided that
the above inequality is satisfied. Rin and Rout MUST also satisfy
the constraint:

$$r <= Rout <= Rin.$$

When several RSpec's, each with rate Rj, j=1,2..., are to be merged
at a split point, the value of Rout is the maximum over all the rates
Rj, and the value of Sout is the minimum over all the slack terms Sj.

   NOTE: The various TSpec functions described above are used by
   applications which desire to combine TSpecs. It is important to
   observe, however, that the properties of the actual reservation
   are determined by combining the TSpec with the RSpec rate (R).

   Because the guaranteed reservation requires both the TSpec and the
   RSpec rate, there exist some difficult problems for shared
   reservations in RSVP, particularly where two or more source
   streams meet. Upstream of the meeting point, it would be
   desirable to reduce the TSpec and RSpec to use only as much
   bandwidth and buffering as is required by the individual source's
   traffic. (Indeed, it may be necessary if the sender is
   transmitting over a low bandwidth link).

   However, the RSpec's rate is set to achieve a particular delay
   bound (and is notjust a function of the TSpec), so changing the
   RSpec may cause the reservation to fail to meet the receiver's
   delay requirements. At the same time, not adjusting the RSpec
   rate means that "shared" RSVP reservations using guaranteed
   service will fail whenever the bandwidth available at a particular
   link is less than the receiver's requested rate R, even if the
   bandwidth is adequate to support the number of senders actually
   using the link. At this time, this limitation is an open problem
   in using the guaranteed service with RSVP.

Guidelines for Implementors

   This section discusses a number of important implementation issues in
   no particular order.

   It is important to note that individual subnetworks are network
   elements and both routers and subnetworks MUST support the guaranteed
   service model to achieve guaranteed service. Since subnetworks
   typically are not capable of negotiating service using IP-based
   protocols, as part of providing guaranteed service, routers will have
   to act as proxies for the subnetworks they are attached to.

   In some cases, this proxy service will be easy. For instance, on
   leased line managed by a WFQ scheduler on the upstream node, the
   proxy need simply ensure that the sum of all the flows' RSpec rates
   does not exceed the bandwidth of the line, and needs to advertise the
   rate-based and non-rate-based delays of the link as the values of C
   and D.

In other cases, this proxy service will be complex. In an ATM
network, for example, it may require establishing an ATM VC for the
flow and computing the C and D terms for that VC. Readers may
observe that the token bucket and peak rate used by guaranteed
service map directly to the Sustained Cell Rate, Burst Size, and Peak
Cell Rate of ATM's Q.2931 QoS parameters for Variable Bit Rate
traffic.

The assurance that datagrams will not be lost is obtained by setting
the router buffer space B to be equal to the token bucket b plus some
error term (described below).

Another issue related to subnetworks is that the TSpec's token bucket
rates measure IP traffic and do not (and cannot) account for link
level headers. So the subnetwork network elements MUST adjust the
rate and possibly the bucket size to account for adding link level
headers. Tunnels MUST also account for the additional IP headers
that they add.

For datagram networks, a maximum header rate can usually be computed
by dividing the rate and bucket sizes by the minimum policed unit.
For networks that do internal fragmentation, such as ATM, the
computation may be more complex, since one MUST account for both
per-fragment overhead and any wastage (padding bytes transmitted) due
to mismatches between datagram sizes and fragment sizes. For
instance, a conservative estimate of the additional data rate imposed
by ATM AAL5 plus ATM segmentation and reassembly is

$$((r/48)*5)+((r/m)*(8+52))$$

which represents the rate divided into 48-byte cells multiplied by
the 5-byte ATM header, plus the maximum datagram rate (r/m)
multiplied by the cost of the 8-byte AAL5 header plus the maximum
space that can be wasted by ATM segmentation of a datagram (which is
the 52 bytes wasted in a cell that contains one byte). But this
estimate is likely to be wildly high, especially if m is small, since
ATM wastage is usually much less than 52 bytes. (ATM implementors
should be warned that the token bucket may also have to be scaled
when setting the VC parameters for call setup and that this example
does not account for overhead incurred by encapsulations such as
those specified in RFC 1483).

To ensure no loss, network elements will have to allocate some
buffering for bursts. If every hop implemented the fluid model
perfectly, this buffering would simply be b (the token bucket size).
However, as noted in the discussion of reshaping earlier,
implementations are approximations and we expect that traffic will
become more bursty as it goes through the network. However, as with

shaping the amount of buffering required to handle the burstiness is
bounded by b+Csum+Dsum*R. If one accounts for the peak rate, this
can be further reduced to

$$M + (b-M)(p-X)/(p-r) + (Csum/R + Dsum)X$$

where X is set to r if (b-M)/(p-r) is less than Csum/R+Dsum and X is
R if (b-M)/(p-r) is greater than or equal to Csum/R+Dsum and p>R;
otherwise, X is set to p. This reduction comes from the fact that
the peak rate limits the rate at which the burst, b, can be placed in
the network. Conversely, if a non-zero slack term, Sout, is returned
by the network element, the buffer requirements are increased by
adding Sout to Dsum.

While sending applications are encouraged to set the peak rate
parameter and reshaping points are required to conform to it, it is
always acceptable to ignore the peak rate for the purposes of
computing buffer requirements and end-to-end delays. The result is
simply an overestimate of the buffering and delay. As noted above,
if the peak rate is unknown (and thus potentially infinite), the
buffering required is b+Csum+Dsum*R. The end-to-end delay without
the peak rate is b/R+Ctot/R+Dtot.

The parameter D for each network element SHOULD be set to the maximum
datagram transfer delay variation (independent of rate and bucket
size) through the network element. For instance, in a simple router,
one might compute the difference between the worst case and best case
times it takes for a datagram to get through the input interface to
the processor, and add it to any variation that may occur in how long
it would take to get from the processor to the outbound link
scheduler (assuming the queueing schemes work correctly).

For weighted fair queueing in a datagram environment, D is set to the
link MTU divided by the link bandwidth, to account for the
possibility that a packet arrives just as a maximum-sized packet
begins to be transmitted, and that the arriving packet should have
departed before the maximum-sized packet. For a frame-based, slotted
system such as Stop and Go queueing, D is the maximum number of slots
a datagram may have to wait before getting a chance to be
transmitted.

Note that multicasting may make determining D more difficult. In
many subnets, ATM being one example, the properties of the subnet may
depend on the path taken from the multicast sender to the receiver.
There are a number of possible approaches to this problem. One is to

choose a representative latency for the overall subnet and set D to
the (non-negative) difference from that latency. Another is to
estimate subnet properties at exit points from the subnet, since the
exit point presumably is best placed to compute the properties of its
path from the source.

    NOTE: It is important to note that there is no fixed set of rules
    about how a subnet determines its properties, and each subnet
    technology will have to develop its own set of procedures to
    accurately compute C and D and slack values.

D is intended to be distinct from the latency through the network
element. Latency is the minimum time through the device (the speed
of light delay in a fiber or the absolute minimum time it would take
to move a packet through a router), while parameter D is intended to
bound the variability in non-rate-based delay. In practice, this
distinction is sometimes arbitrary (the latency may be minimal) -- in
such cases it is perfectly reasonable to combine the latency with D
and to advertise any latency as zero.

    NOTE: It is implicit in this scheme that to get a complete
    guarantee of the maximum delay a packet might experience, a user
    of this service will need to know both the queueing delay
    (provided by C and D) and the latency. The latency is not
    advertised by this service but is a general characterization
    parameter (advertised as specified in [8]).

    However, even if latency is not advertised, this service can still
    be used. The simplest approach is to measure the delay
    experienced by the first packet (or the minimum delay of the first
    few packets) received and treat this delay value as an upper bound
    on the latency.

The parameter C is the data backlog resulting from the vagaries of
how a specific implementation deviates from a strict bit-by-bit
service. So, for instance, for datagramized weighted fair queueing, C
is set to M to account for packetization effects.

If a network element uses a certain amount of slack, S_i, to reduce
the amount of resources that it has reserved for a particular flow,
i, the value S_i SHOULD be stored at the network element.
Subsequently, if reservation refreshes are received for flow i, the
network element MUST use the same slack S_i without any further
computation. This guarantees consistency in the reservation process.

As an example for the use of the slack term, consider the case where
the required end-to-end delay, Dreq, is larger than the maximum delay
of the fluid flow system. The latter is obtained by setting R=r in
the fluid delay formula (for stability, R>=r must be true), and is
given by

$$b/r + Ctot/r + Dtot.$$

In this case the slack term is

$$S = Dreq - (b/r + Ctot/r + Dtot).$$

The slack term may be used by the network elements to adjust their
local reservations, so that they can admit flows that would otherwise
have been rejected. A network element at an intermediate network
element that can internally differentiate between delay and rate
guarantees can now take advantage of this information to lower the
amount of resources allocated to this flow. For example, by taking an
amount of slack s <= S, an RCSD scheduler [5] can increase the local
delay bound, d, assigned to the flow, to d+s. Given an RSpec, (Rin,
Sin), it would do so by setting Rout = Rin and Sout = Sin - s.

Similarly, a network element using a WFQ scheduler can decrease its
local reservation from Rin to Rout by using some of the slack in the
RSpec. This can be accomplished by using the transformation rules
given in the previous section, that ensure that the reduced
reservation level will not increase the overall end-to-end delay.

Evaluation Criteria

   The scheduling algorithm and admission control algorithm of the
   element MUST ensure that the delay bounds are never violated and
   datagrams are not lost, when a source's traffic conforms to the
   TSpec. Furthermore, the element MUST ensure that misbehaving flows
   do not affect the service given to other flows. Vendors are
   encouraged to formally prove that their implementation is an
   approximation of the fluid model.

Examples of Implementation

   Several algorithms and implementations exist that approximate the
   fluid model. They include Weighted Fair Queueing (WFQ) [2], Jitter-
   EDD [3], Virtual Clock [4] and a scheme proposed by IBM [5]. A nice
   theoretical presentation that shows these schemes are part of a large
   class of algorithms can be found in [6].

Examples of Use

   Consider an application that is intolerant of any lost or late
   datagrams. It uses the advertised values Ctot and Dtot and the TSpec
   of the flow, to compute the resulting delay bound from a service
   request with rate R. Assuming R < p, it then sets its playback point
   to [(b-M)/R*(p-R)/(p-r)]+(M+Ctot)/R+Dtot.

Security Considerations

   This memo discusses how this service could be abused to permit denial
   of service attacks. The service, as defined, does not allow denial
   of service (although service may degrade under certain
   circumstances).

Appendix 1: Use of the Guaranteed service with RSVP

   The use of guaranteed service in conjunction with the RSVP resource
   reservation setup protocol is specified in reference [9]. This
   document gives the format of RSVP FLOWSPEC, SENDER_TSPEC, and ADSPEC
   objects needed to support applications desiring guaranteed service
   and gives information about how RSVP processes those objects. The
   RSVP protocol itself is specified in Reference [10].

References

   [1] Shenker, S., and J. Wroclawski, "Network Element Service
   Specification Template", RFC 2216, September 1997.

   [2] A. Demers, S. Keshav and S. Shenker, "Analysis and Simulation of
   a Fair Queueing Algorithm," in Internetworking: Research and
   Experience, Vol 1, No. 1., pp. 3-26.

   [3] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for
   Packet Switching Networks," in Proc. ACM SIGCOMM '90, pp. 19-29.

   [4] D. Verma, H. Zhang, and D. Ferrari, "Guaranteeing Delay Jitter
   Bounds in Packet Switching Networks," in Proc. Tricomm '91.

   [5] L. Georgiadis, R. Guerin, V. Peris, and K. N. Sivarajan,
   "Efficient Network QoS Provisioning Based on per Node Traffic
   Shaping," IBM Research Report No. RC-20064.

   [6] P. Goyal, S.S. Lam and H.M. Vin, "Determining End-to-End Delay
   Bounds in Heterogeneous Networks," in Proc. 5th Intl. Workshop on
   Network and Operating System Support for Digital Audio and Video,
   April 1995.

    [7] A.K.J. Parekh, A Generalized Processor Sharing Approach to Flow
    Control in Integrated Services Networks, MIT Laboratory for
    Information and Decision Systems, Report LIDS-TH-2089, February 1992.

    [8] Shenker, S., and J. Wroclawski, "General Characterization
    Parameters for Integrated Service Network Elements", RFC 2215,
    September 1997.

    [9] Wroclawski, J., "Use of RSVP with IETF Integrated Services", RFC
    2210, September 1997.

    [10] Braden, R., Ed., et. al., "Resource Reservation Protocol (RSVP)
    - Version 1 Functional Specification", RFC 2205, September 1997.

Authors' Addresses

    Scott Shenker
    Xerox PARC
    3333 Coyote Hill Road
    Palo Alto, CA 94304-1314

    Phone: 415-812-4840
    Fax: 415-812-4471
    EMail: shenker@parc.xerox.com

    Craig Partridge
    BBN
    2370 Amherst St
    Palo Alto CA 94306

    EMail: craig@bbn.com

    Roch Guerin
    IBM T.J. Watson Research Center
    Yorktown Heights, NY 10598

    Phone: 914-784-7038
    Fax: 914-784-6318
    EMail: guerin@watson.ibm.com