Internet Engineering Task Force I. Stoica UC, Berkeley
Internet Draft H. Zhang CMU
Expires April 2003 N. Venkitaraman Motorola Labs
J. Mysore Motorola Labs

October 2002

Per Hop Behaviors Based on Dynamic Packet State

<draft-stoica-diffserv-dps-02.txt>

Status of this Memo

Copyright Notice

Abstract

This document proposes a family of Per-Hop Behaviors (PHBs) based on Dynamic Packet State (DPS) in the context of the differentiated service architecture. With these PHBs, distributed algorithms can be devised to implement services with flexibility, utilization, and assurance levels similar to those that can be provided with per-flow mechanisms.

With Dynamic Packet State, each packet carries in its header, in addition to the PHB codepoint, some PHB-specific state. The state is initialized by the ingress node. Interior nodes process each incoming packet based on the state carried in the packet's header, updating both its internal state and the state in the packet's header before forwarding it to the next hop. By using DPS to coordinate actions of edge and interior nodes along the path traversed by a flow, distributed algorithms can be designed

to approximate the behavior of a broad class of "stateful"
networks using networks in which interior nodes do not maintain

per-flow state. We give examples of services that can be implemented by PHBs based on DPS. We also discuss several possible solutions for encoding Dynamic Packet State that have the minimum incompatibility with IPv4.

1. Introduction

While the diffserv architecture [Blake98] is highly scalable, the services it can provide have lower flexibility, utilization, or assurance levels than services provided by architectures that employ per-flow management at every node. It is debatable whether this should be of significant concern. For example, the low utilization by the premium traffic may be acceptable if the majority of traffic will be best effort, either because the best effort service is "good enough" for majority applications or the price difference between premium traffic and best effort traffic is too high to justify the performance difference between them. Alternatively, if the guaranteed nature of service assurance is not needed, i.e., statistical service assurance is sufficient for premium service, we can then achieve higher network utilization. Providing meaningful statistical service is still an open research problem. A discussion of these topics is beyond the scope of this document. Furthermore, there is usually a tradeoff between the complexity of the QoS mechanism and the efficiency of the resource usage. While intserv-style guaranteed service can achieve high resource utilization, premium service needs a much simpler mechanism.

This document proposes a new set of PHBs based on Dynamic Packet State (DPS). These PHBs have relative low complexities (do not require per-flow state), but can be used to implement distributed algorithms to provide services with flexibility, utilization, and assurance levels similar to those that can be achieved with per-flow mechanisms. DS domains implemented with these type of PHBs should interoperate with DS domains implemented with other PHBs.

With Dynamic Packet State, each packet carries in its header, in addition to the PHB codepoint, some PHB-specific state. The state is initialized by an ingress node, when the packet enters the DS domain. Interior nodes process each incoming packet based on the state carried in the packet's header, updating both its internal state and the state in the packet's header before forwarding it. By using DPS to coordinate actions of edge and interior nodes along the path traversed by a flow, distributed algorithms can be designed to approximate the behavior of a broad class of "stateful" networks. Consequently, introducing PHBs based on DPS will significantly increase the flexibility and capabilities of the services that can be built within the diffserv architecture. In particular, we will show that a variety of QoS services can be implemented by PHBs based on DPS. These include weighted fair

share service, and distributed admission control service.

In this document, we use flow to refer to a subset of packets
that traverse the same path inside a DS domain between two edge

nodes. Thus, with the highest level of traffic aggregation, a flow consists of all packets between the same pair of ingress and egress nodes.

This document is organized as follows. Section 2 gives a general description of PHBs based on DPS. Section 3 presents several services that can be implemented with PHBs based on DPS. Section 4 discusses alternative techniques of storing state in the packet's header. Sections 5 briefly discusses some issues related to the specification of DPS PHB's, such as codepoints, tunneling behavior, and interaction with other PHB's. Section 6 discusses security issues.

2. Description of Per-Hop Behaviors Based on Dynamic Packet State

Unlike common PHB codepoints [Blake98, Heinanen99, Jacobson98], a PHB codepoint based on DPS has extra state associated with it. This state is initialized by ingress nodes and carried by packets inside the DS domain. The state semantic is PHB dependent. The values taken by the state can be either flow, path, or packet dependent. The state carried by packets can be used by interior nodes for a variety of purposes such as, packet scheduling, updating the local node's state, or extending the codepoint space.

When a PHB based on DPS is defined, in addition to the guidelines given in [Blake98], the following items must be specified:

- state semantic - the meaning of the information carried by the packets
- state placement - where is the information stored in the packet's header
- encoding format - how is the information encoded in packets

For example, consider a PHB that implements the Stateless Prioritized Fair Queue Sharing algorithm, which is described in Section 3.1. In this case, the state carried by a packet is based on an estimate of the current rate of the flow to which the packet belongs. The state can be placed either (a) between layer two and layer three headers, (b) as an IP option, or (c) in the IP header (see Section 4). Finally, the rate can be encoded by using a floating point like format as described in Section 4.1.1.

In addition, the following requirement, called the transparency requirement, must be satisfied

- All changes performed at ingress nodes or within the DS

domain on packets' headers (possible for the purpose of
the state) must be undone by egress nodes

Any document defining a PHB based on DPS must specify a default placement of the state in the packet header and a default bit encoding format. However, to increase the flexibility, it is acceptable for documents to define alternate state placements and encoding formats. Any router that claims to be compatible with a particular PHB based on DPS must support at least the default placement and the default bit encoding format.

3. Examples of Services that can be Implemented by PHBs Based on DPS

To illustrate the power and the flexibility of the PHBs based on DPS, we give a few examples. In the first, we address the congestion control problem by approximating the functionality of a "reference" network in which each node performs fair queuing.

3.1. Stateless Prioritized Fair Queue-sharing (SPFQ)

We first explain SPFQ using an idealized fluid model and then present its packetized version.

3.1.1 Fluid Model Algorithm: Bit Labeling

We first restate the key observation in CSFQ [Stoica98] that we will also use. In a router implementing fair queuing, all flows that are bottlenecked at a router have the same output rate. We call this the rate threshold(r_t(t)).

Let C be the capacity of an output link at a router, and r_i(t) the arrival rate of flow i in bits per second. Let A(t) denote the total arrival rate of n flows: A(t)= r_1(t) + r_2(t) + ... + r_n(t). If A(t) <= C then no bits are dropped. But if A(t) > C, then r_t(t) is the unique solution to C = min(r_1(t), r_t(t)) + min(r_2(t), r_t(t)) + ... + min(r_n(t), r_t(t)).

In general, if max-min bandwidth allocations are achieved, each flow i receives service at a rate given by min(r_i(t), r_t(t)).

For every flow, in any given second, we consider up to r_t(t) bits of the flow as being conforming, and all bits in excess of that as being non-conforming. If we mark every bit of a flow as being conforming or non-conforming, we can obtain the allocation provided by a fair queuing router, by simply having routers accept all conforming bits of a flow and dropping all non-conforming bits.

What we need now, is a labeling algorithm at the ingress node, that would enable a router to distinguish between conforming and non-conforming traffic. Consider the simple sequential labeling algorithm:

label(bit)

```
   served += 1
   bit->label = served
```
where the value of 'served' is reset to 0, after every second.

Let us suppose that the rate at which each flow is sending bits is constant. The result of this algorithm is that during any given second, the bits from a flow sending at rate r_i bits per second are marked sequentially from 1 to r_i; and the label is reset to 0 at the end of each second. Then, for any given flow, accepting bits with label 1 to 'r_a', would be equivalent to providing the flow with a rate of 'r_a' bits per second. So, if all bits carry such a label, a router can simply identify non-conforming bits to be those with label > r_t and drop them. Consequently, no flow can receive a service in excess of r_t . Furthermore, as all bits with label <= r_t are accepted, all flows sending at a rate less than or equal to r_t will not have any of their bits dropped.

As described in [Venkitar02], a key advantage of such a labeling procedure is that it allows us to convey rate information as well as intra-flow priority using the same field in the packet header.

3.1.2 Packet Labeling

In order to extend the sequential labeling algorithm given for the fluid model to a packetized model, we essentially need to take variable packet sizes into account. Hence, instead of incrementing the counter 'served' by 1 (which is the size of any packet in a network with purely single-bit packets), we increment the value of 'served' by the size of the packet. Given below is a pseudo code for the packetized version of the sequential marking algorithm.

```
label(pkt)
  served += pkt->size
  pkt->label = served
```

where the value of served is reset to 0, after a fixed size epoch.

All ingress routers in a DS domain must use epochs of equal duration. The size of the epoch is a design parameter that should be chosen to reflect a tradeoff between mimicking the fluid model accurately and not giving an unfair advantage to flows that arrived most recently in the system. To understand this tradeoff, suppose that the epoch is chosen to be very long and that a new flow arrives in the middle of an epoch. Then the bits from the new flow would be labeled starting from a value of one and would have a higher priority throughout the rest of the epoch than the bits of flows that have been sending bits from the beginning of the epoch.

3.1.3 Forwarding Decision in a Router

[Stoica98], [Cao00], [Barnes01] and [Venkitar02] discuss algorithms for updating the rate threshold (r_t) based on link state, i.e,

based on parameters such as queue length and the aggregate accepted rate of packets. The forwarding decision in a router is then made based on the following algorithm:

```
enque(pkt)
  if (pkt->label <= r_t)
   Accept(pkt)
  else
   Drop(pkt)
```

3.1.4 Additional Services based on SPFQ

We now present examples of labeling methods at the edge that can provide different services while retaining the same forwarding behavior.

3.1.4.1 Weighted SPFQ

The SPFQ algorithm can be readily extended to support flows with different weights. Let w_i be the weight of flow i. An allocation is weighted fair if all bottlenecked flows have the same value for r_i/w_i. The only major change required to achieve weighted fair allocation is in the ingress labeling algorithm, where we need to use 'served/w_i' instead of 'served'. This enables per-flow service differentiation without maintaining per-flow state in the core nodes of the network.

3.1.4.2 Minimum bandwidth allocation

From the forwarding algorithm given in section 2, it is clear that the packets marked with lower values of label are dropped only after all packets with larger labels have been dropped. This suggests that packets marked with the smallest label (of 0) will not be dropped as long as the aggregate rate of such packets does not exceed the link capacity. So, for a flow requiring a minimum bandwidth allocation of 'b_min', labeling packets with the smallest label at a rate of 'b_min' would ensure that the flow will receive the rate that it has been guaranteed within a reasonable time window (assuming that there is no packet loss due to channel error). An admission control mechanism should be used to ensure that the aggregate reserved rate does not exceed the capacity of the link. A distributed admission control mechanism, such as the one proposed in section 3.2 can be used for this purpose.

3.2 Distributed Admission Control

The previous examples focused on data path mechanisms and services. In this section, we will show that PHBs based on DPS can also implement control plane services such as distributed admission control.

Admission control is a central component in providing quantitatively defined QoS services. The main job of the admission control test is to ensure that the network resources are not over-committed. In

particular it has to ensure that the sum of the reservation rates of all flows that traverse any link in the network is no larger than the link capacity C.

A new reservation request is granted if it passes the admission test at each hop along its path. There are two main approaches to implementing admission control. Traditional reservation-based networks adopt a distributed architecture in which each node is responsible for its local resources, and where nodes are assumed to maintain per-flow state. To support the dynamic creation and deletion of fine grained flows, a large quantity of dynamic per flow state needs to be maintained in a distributed fashion. Complex signaling protocols (e.g., RSVP and ATM UNI) are used to maintain the consistency of this per-flow state.

A second approach is to use a centralized bandwidth broker that maintains the topology as well as the state of all nodes in the network. In this case, the admission control can be implemented by the broker, eliminating the need for maintaining distributed state. Such a centralized approach is more appropriate for an environment where most flows are long lived, and set-up and tear-down events are rare. However, to support fine grained and dynamic flows, there may be a need for a distributed broker architecture, in which the broker database is replicated or partitioned. Such an architecture eliminates the need for a signaling protocol, but requires another protocol to maintain the consistency of the different broker databases. Unfortunately, since it is impossible to achieve perfect consistency, this may create race conditions and/or resource fragmentation.

A third approach is to use a simplified provisioning model that is not aware of the details of the network topology, but instead admits a new flow if there is sufficient bandwidth available for the flow's packets to travel anywhere in the network with adequate QoS. This simplified model may be based on static provisioning and service level agreements, or on a simple dynamic bandwidth broker. In any case, the tradeoff made in return for the simplicity is that the admission control decision must be more conservative, and a much smaller level of QoS-controlled service can be supported.

In the following, we show that by using a PHB based on DPS, it is possible to implement distributed admission control for guaranteed services in a DS domain. In our scheme, for each reservation, the ingress node generates a request message that is forwarded along the path. In turn, each interior node decides whether or not to accept the request. When the message reaches the egress node it is returned to the ingress, which makes the final decision. We do not make any reliability assumptions about the request messages. In addition, the algorithms does not require reservation termination messages. In the following we describe the per-hop admission control. [StoZha99] describes how this scheme can be integrated with RSVP to provide end-to-end delay bounded services.

3.2.1. Per-Hop Admission Control

The solution is based on two main ideas. First, we always maintain a conservative upper bound of the aggregate reservation R, denoted

R_bound, which we use for making admission control decisions. R_bound is updated with a simple rule:

R_bound = R_bound + r

whenever a request for a rate r is received and R_bound + r <= C. It should be noted that in order to maintain the invariant that R_bound is an upper bound of R, this algorithm does not need to detect duplicate request messages, generated either due to retransmission in case of packet loss or retry in case of partial reservation failures. Of course, the obvious problem with the algorithm is that R_bound will diverge from R. At the limit, when R_bound reaches the link capacity C, no new requests can be accepted even though there might be available capacity.

To address this problem, we introduce a separate algorithm, based on DPS, that periodically estimates the aggregate reserved rate. Based on this estimate we compute a second upper bound for R, and then use it to re-calibrate the upper bound R_bound. An important aspect of the estimation algorithm is that it can be actually shown that the discrepancy between the upper bound and the actual reserved rate R is bounded. Then the re-calibration process reduces to choosing the minimum of the two upper bounds.

3.2.2. Estimation Algorithm for the Aggregate Reservation

To estimate the aggregate reservation, denoted R_est, we again use DPS. In this case, the state of each packet consists of the amount of bits a flow is entitled to send during the interval between the time when the previous packet was transmitted up to the time when the current packet is transmitted. Note that unlike the previous examples, in this case the state carried by the packet does not affect the packet's processing by interior nodes. This state is solely used to compute each node's aggregate reservation.

The estimation performed by interior nodes is based on the following simple observation: the sum of state values of packets of all flows received during an interval (a, a + T_W] is a good approximation for the total number of bits that all flows are entitled to send during this interval. Dividing this sum by T_W, we obtain the aggregate reservation rate. This is basically the rate estimation algorithm, though we need to account for several estimation errors. In particular, we need to account for the fact that not all reservations continue for the entire duration of interval (a, a + T_W].

We divide time into intervals of length T_W. Let (u1, u2] be such an interval, where u2 = u1 + T_W. Let T_I be the maximum inter-departure time between two consecutive packets in the same flow and T_J be the maximum jitter of a flow, both of which are much smaller than T_W. Further, each interior node is associated

a global variable Ra which is initialized at the beginning of each interval (u1, u2] to zero, and is updated to Ra + r every time a request for a reservation r is received and the admission test is passed, i.e., R_bound + r <= C.

Let Ra(t) denote the value of this variable at time t. Since interior nodes do not differentiate between the original and duplicate requests, Ra(t) is an upper-bound of the sum of all reservations accepted during the interval (u1, t]. (For simplicity, here we assume that a flow which is granted a reservation during the interval (u1, u2] becomes active no later than u2.) Then, it can be shown that the aggregate reservation at time u2, R(u2), is bounded by

$$R(u2) < R\_est(u2)/(1-f) + Ra(u2), \quad (7)$$

where f = (T_I + T_J)/T_W. Finally, this bound is used to re-calibrate the upper bound of the aggregate reservation R_bound(u1) as follows

$$R\_bound(u2) = \min(R\_bound(u2), R\_est(u2)/(1-f) + Ra(u2)). \quad (8)$$

Figure 1 shows the pseudocode of the admission decision and of the aggregate reservation estimation algorithm at ingress and interior nodes. We make several observations. First, the estimation algorithm uses only the information in the current interval. This makes the algorithm robust with respect to loss and duplication of signaling packets since their effects are "forgotten" after one time interval. As an example, if a node processes both the original and a duplicate of the same reservation request during the interval (u1, u2], R_bound will be updated twice for the same flow. However, this erroneous update will not be reflected in the computation of R_est(u3), since its computation is based only on the state values received during (u2, u3]. As a consequence, it can be show that the admission control algorithm can asymptotically reach a link utilization of C (1 - f)/(1 + f) [StoZha99].

A possible optimization of the admission control algorithm is to add reservation termination messages. This will reduce the discrepancy between the upper bound R_bound and the aggregate reservation R. However, in order to guarantee that R_bound remains an upper bound for R, we need to ensure that a termination message is sent at most once, i.e., there are no retransmissions if the message is lost. In practice, this property can be enforced by edge nodes, which maintain per-flow state.

To ensure that the maximum inter-departure time is no larger than T_I, the ingress node may need to send a dummy packet in the case when no data packet arrives for a flow during an interval T_I. This can be achieved by having the ingress node to maintain a timer with each flow. An optimization would be to aggregate all "micro-flows" between each pair of ingress and egress nodes into one flow, and compute the state value based on the aggregated reservation rate,

and insert a dummy packet only if there is no data packet for the aggregate flow during an interval.

Figure 1 - Admission control and rate estimation algorithm.

```
-------------------------------------------------------------------
                          Ingress node
-------------------------------------------------------------------
   on packet p departure:
     i = get_flow(p);
     state(p) <- r[i] * (crt_time - prev_time[i]);
     prev_time[i] = crt_time;
-------------------------------------------------------------------
                           Interior node
-------------------------------------------------------------------
    Reservation Estimation         |     Admission Control
-------------------------------------------------------------------
   on packet p arrival:            |  on reservation request r:
     b <- state(p);                |    /* admission ctrl. test */
     L = L + b;                    |    if (R_bound + r <= C)
                                   |      Ra = Ra + r;
   on time-out T_W:                |      R_bound = R_bound + r;
     /* estimated reservation */   |      accept(r);
     R_est = L / T_W;              |    else
     R_bound = min(R_bound,        |      deny(r);
               R_est/(1 - f) + Ra);|
     Ra = 0;                       |
 -------------------------------------------------------------------
```

4. Carrying State in Packets

There are at least three ways to encode state in the packet header: (1) introduce a new IP option, and insert the option at the ingress router, (2) introduce a new header between layer 2 and layer 3, and (3) use the existing IP header.

Option number 23 has been assigned for adding DPS state in packets. Inserting an IP option, has the potential to provide a large space for encoding state. However it will require all routers within a DS domain to process IP options, which could complicate packet processing.

Introducing a new header between layer 2 and layer 3 would require solutions be devised for different layer 2 technologies. In the context of MPLS [Rosen98, Rosen99] networks, the state can be encoded in a special label. One way to do this is by using a particular encoding of the experimental use field indicating a nested label on the label stack that carried the PHB-specific state information rather than an ordinary label. In this case, the label on the top of the stack would indicate the label-switched path, and the inner label the PHB-specific state. This would require a small

addition to the MPLS architecture to allow two labels to be pushed or popped in unison, and treated as a single entity on the label stack.

4.1. Encoding State within an IP header

In this section, we discuss the third option: storing the additional states in the IP header. The biggest problem with using the IP header is to find enough space to insert the extra information. The main challenge is to remain fully compatible with current standards and protocols. In particular, we want the network domain to be transparent to end-to-end protocols, i.e., the egress node should restore the fields changed by ingress and interior nodes to their original values. In this respect, we observe that there is an ip_off field in the IPv4 header to support packet fragmentation/reassembly which is rarely used. For example, by analyzing the traces of over 1.7 million packets on an OC-3 link [nlanr], we found that less than 0.22% of all packets were fragments. In addition, ther are a relatively small number of distinct fragment sizes. Therefore, it is possible to use a fraction of ip_off field to encode the fragment sizes, and the remaining bits to encode DPS information. The idea can be implemented as follows. When a packet arrives at an ingress node, the node checks whether a packet is a fragment or needs to be fragmented. If neither of these is true, all 13 bits of the ip_off field in the packet header will be used to encode DPS values. If the packet is a fragment, the fragment size is recoded into a more efficient representation and the rest of the bits is used to encode the DPS information. The fragment size field will be restored at the egress node.

In the above, we make the implicit assumption that a packet can be fragmented only by ingress nodes, and not by interior nodes. This is consistent with the diffserv view that the forwarding behavior of a network's component is engineered to be compatible throughout a domain.

In summary, this gives us up to 13 bits in the current IPv4 header to encode the PHB specific state.

4.2. Example of State Encoding

The state encoding is PHB dependent. In this section, we give examples of encoding the state for the services described in Section 3.

4.2.1. Encoding Flow's Rate

Recall that in SPFQ, the PHB state is determined by the current rate of the flow to which the packet belongs. One possible way to represent the rate estimate is to restrict it to only a small number of possible values. For example if we limit it to 128 values, only seven bits are needed to represent this rate. While this can be a reasonable solution in practice, in the following we propose a more sophisticated representation that allows us to express a larger range of values.

Let r denote the packet label. In the most general case r could be a floating poing number. To represent r we use an m bit mantissa and an n bit exponent. Since r >= 0, it is possible to gain

an extra bit for mantissa. For this we consider two cases:
(a) if r >= 2^m we represent r as the closest value of the form u 2^v, where 2^m <= u <= 2^(m+1). Then, since the (m+1)-th most significant bit in the u's representation is always 1, we can ignore it. As an example, assume m = 3, n = 4, and r = 19 = 10011. Then 19 is represented as 18 = u*2^v, where u = 9 = 1001 and v = 1. By ignoring the first bit in the representation of u the mantissa will store 001, while the exponent will be 1.
(b) On the other hand, if r < 2^m, the mantissa will contain r, while the exponent will be 2^n - 1. For example, for m = 3, n = 4, and r = 6 = 110, the mantissa is 110, while the exponent is 1111. Converting from one format to another can be efficiently implemented. Figure 2 shows the conversion code in C. For simplicity, here we assume that integers are truncated rather than rounded when represented in floating point.

Figure 2. The C code for converting between integer and floating point formats. m represents the number of bits used by the mantissa; n represents the number of bits in the exponent.

```
  intToFP(int val, int *mantissa, int *exponent) {
    int nbits = get_num_bits(val);
    if (nbits <= m) {
      *mantissa = val;
      *exponent = (1 << n) - 1;
    } else {
      *exponent = nbits - m - 1;
      *mantissa = (val >> *exponent) - (1 << m);
    }
  }

  FPToInt(int mantissa, int exponent) {
    int tmp;
    if (exponent == ((1 << n) - 1))
      return mantissa;
    tmp = mantissa | (1 << m);
    return (tmp << exponent)
  }
```

By using m bits for mantissa and n for exponent, we can represent any integer in the range [0..(2^(m+1)-1) * 2^(2^n - 1)] with a relative error bounded by (-1/2^(m+1), 1/2^(m+1)). For example, with 7 bits, by allocating 3 for mantissa and 4 for exponent, we can represent any integer in the range [1..15*2^15] with a relative error of (-6.25%, 6.25%). The worst relative error case occurs when the mantissa is 8. For example the number r = 271 = 100001111 is encoded

as u = 1000, v=5, with a relative error of (8*2^5 - 271)/271 = -0.0554 = -5.54%. Similarly, r = 273 = 100010001 is encoded as u = 1001, v = 5, with a relative error of 5.55%.

4.2.2. Encoding Reservation State

As shown in Figure 1, when estimating the aggregate reservation, the PHB state represents the number of bits that a flow is entitled to send during the interval between the time when the previous packet of the flow has been transmitted until the current packet is transmitted. This number can be simply encoded as an integer b. To reduce the range, a possibility is to store b/l instead of b, where l is the length of the packet.

4.3. Encoding Multiple Values

Since the space in the packet's header is a scarce resource, encoding multiple values is particularly challenging. In this section we discuss two general methods that helps to alleviate this difficulty.

In the first method, the idea is to leverage additional knowledge about the state semantic to achieve efficient encoding. In particular one value can be stored as a function of other values. For example, if a value is known to be always greater than the other values, the larger value can be represented in floating point format, while the other values may be represented as fractions of this value.

The idea of the second method is to have different packets within a flow carry different state formats. This method is appropriate for PHBs that do not require all packets of a flow to carry the same state. For example, in estimating the aggregate reservation (see Section 3.2) there is no need for every packet to carry the number of bits the flow is entitled to send between the current time and the time when the previous packet has been transmitted. The only requirement is that the distance between any two consecutive packets that carry such values to be no larger than T_I. Other packets in between can carry different information. Similarly, if we encode the IP fragment size in the packet's state, the packet has to carry this value only if the IP fragment is not zero. When the IP fragment is zero the packet can carry other state instead. On the other hand, note that in SPFQ, it is mandatory that every packet be labelled by the ingress edge, as this value is used in making forward/drop decisions by ingress routers.

5. Specification Issues

This section briefly describes some issues related to drafting specifications for PHB's based on DPS.

5.1. Recommended Codepoints

At this time it is appropriate to use values drawn from the 16

codepoints [Nichols98] reserved for local and experimental use (xxxx11) to encode PHBs based on DPS.

5.2. Interaction with other PHBs

The interaction of DPS PHB's with other PHB's obviously depends on the PHB semantic. It should be noted that the presence of other PHB's in a node may affect the computation and update of DPS state as well as the actual forwarding behavior experienced by the packet.

5.3. Tunneling

When packets with PHBs based on DPS are tunneled, the end-nodes must make sure that (1) the tunnel is marked with a PHB that does not violate the original PHB semantic, and (2) the PHB specific state is correctly updated at the end of the tunnel. This requirement might be met by using a tunnel PHB that records and updates packet state, and then copying the state from the encapsulating packet to the inner packet at the tunnel endpoint. Alternatively, the behavior of the tunnel might be measured or precomputed in a way that allows the encapsulated packet's DPS state to be updated at the decapsulation point without requiring the tunnel to support DPS behavior.

6. Security Considerations

The space allocated for the PHB state in the packet header must be compatible with IPsec. In this context we note that using the fragment offset to carry PHB state does not affect IPsec's end-to-end security, since the fragment offset is not used for cryptographic calculations [Kent98]. Thus, as it is the case with the DS field [Nichols98], IPSec does not provide any defense against malicious modifications of the PHB state. This leaves the door open for theft of service, which inturn May cause denial of service to other conforming users.
For example, in SPFQ, a label based on a small rate estimate may cause disproportionate bandwidth being allocated to the flow inside the DS domain. In the example in Section 3.2.2, the under estimation of the aggregate reservation can lead to resource overprovision.
One way to expose denial of service attacks is by auditing. In this context, we note that associating state with PHBs makes it easier to perform efficient auditing at interior nodes. For example, in SPFQ, an eventual attack can be detected by simply measuring a flow rate and then comparing it against the label carried by the flow's packets.

Security considerations covered in [Blake98] that correspond to diffserv code points also apply to PHB code points for DPS.

7. Conclusions

In this document we have proposed an extension of the diffserv architecture by defining a new set of PHBs that are based on Dynamic Packet State. By using DPS to coordinate actions of edge and interior nodes along the path traversed by a flow, distributed

algorithms can be designed to approximate the behavior of a broad class of "stateful" networks within the diffserv architecture. Such an extension will significantly increase the flexibility and capabilities of the services that can be provided by diffserv.

8. References

[Barnes01] R. Barnes, R. Srikant, J. Mysore, N. Venkitaraman. Analysis of Stateless Fair Queuing Algorithms, Proc. of the 35th Annual Conference on Information Sciences and Systems, March 2001.

[Blake98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services, RFC 2475 December 1998.

[Cao00] Z. Cao, Z Wang and E. Zegura, Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State, Proc. of INFOCOM 2000.

[Heinanen99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group, RFC 2597, June 1999.

[Jacobson98] V. Jacobson, K. Poduri and K. Nichols. An Expedited Forwarding PHB, RFC 2598, June 1999.

[Kent98] S. Kent and R. Atkinson. IP Authentication Header, RFC 2402, November 1998.

[Nichols98] K. Nichols, S. Blake, F. Baker, and D. L. Black. Definition of the Differentiated Services Field (DS Field) in the ipv4 and ipv6 Headers, RFC 2474, December 1998.

[Stoica98] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In Proceedings ACM SIGCOMM'98, pages 118-130, Vancouver, September 1998.

[StoZha99] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per-flow Management. In Proceedings of ACM SIGCOMM'99, Boston, September 1999.

[Venkitar02] N. Venkitaraman, J. Mysore, M. Needham. Core-Stateless Utility Function based Rate Allocation. Proceedings of PfHSN'2002, Berlin, April 2002.

9. Author's Addresses

Ion Stoica
645 Soda Hall
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
istoica@cs.berkeley.edu

Hui Zhang
Wean Hall 7115
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
hzhang@cs.cmu.edu

Narayanan Venkitaraman
Motorola Labs

Jayanth Mysore
Motorola Labs,

1301 E. Algonquin Rd.
Schaumburg, IL 60196
venkitar@labs.mot.com

1301 E. Algonquin Rd.
Schaumburg, IL 60196
jayanth@labs.mot.com

10. Full Copyright Statement