DiffServ Service API

Reference Guide

Control Plane-Platform Development Kit 2.11

March 2004

Contents

Figures

Revision History

Revision	Description	Date	Author
2.11	Updated for Release 2.11	March 2004	Ds Sreedhara
2.1	Updated for Release 2.1	December 2003	Ds Sreedhara
2.0	Updated for Release 2.0	August 2003	Ds Sreedhara

Part 1: Overview

1 Overview

In packet data networks, the quality of the data being carried is gaining attention. The quality can be quantized and qualified using the following set of parameters: throughput, jitter, delay, loss, relative priority of access to network resources, and so on.

The Differentiated Service Mechanism is one way of providing quality service to the data traffic. The DiffServ mechanism involves the following processing elements, the combination of which provides quality of service to the data traffic in a particular direction:

- Classification element: This processing element classifies the data into a particular flow stream.
- Metering element: This element measures incoming classified data traffic for compliance with its agreed traffic parameters.
- Marking element: This element marks the data traffic to indicate a particular PHB.
- Queuing and scheduling element: This element treats the traffic in a such a way that it implements the per hop behavior needed by the data.
- Shaper: This element shapes the traffic so that the burstiness in the data traffic is removed.

Figure 1 depicts the typical architecture or relationship between DiffServ Service Provisioning Applications and NPF APIs.

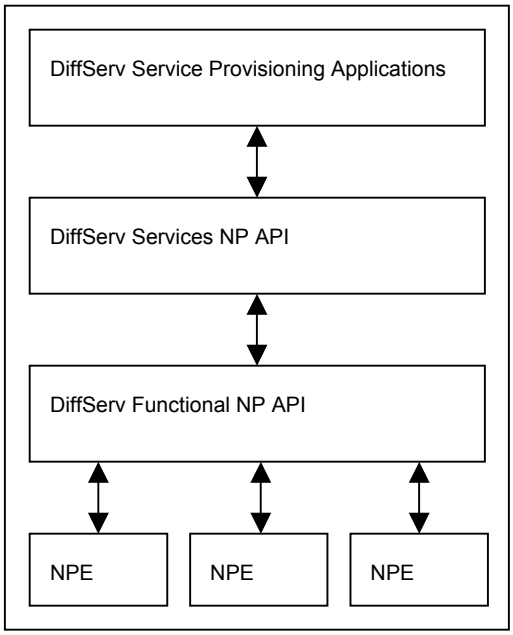

Figure 1. DiffServ service provisioning applications – NP API architectural relationship

The DiffServ Services API provides a generic interface for configuring and managing the forwarding plane of the DiffServ component layer. Service provisioning applications use these APIs to configure and manage the DiffServ information.

The protocol stacks and network processors available from different vendors can be easily integrated with NPF APIs. The APIs included in the Control Plane Platform Development Kit (CP-PDK) are based on NPF APIs. For more information about NPF, refer to http://www.npforum.org/.

1.1 Scope

The DiffServ Service API provides an interface for the control and management of DPE elements. The scope of the API is restricted to the provision of the well-defined PHB, defined as per RFC 2597 and 2598.

1.2 Assumptions

The following assumptions apply to the DiffServ Service APIs and their associated data structures:

- All the DPEs should be configured and managed independently
- Flexibility of associating one DPE with another should be provided
- Multiple entries in the DPE can map into an entry in another DPE. Therefore, it should provide multiple to one association.
- Enabling and disabling of an entry in the DPE should be possible

1.3 Dependencies

The current document does not cover proprietary implementation of any new DPE.

1.4 References

This document depends on the following NPF documents:

- NP Forum – Software API Framework Lexicon Implementation Agreement Revision 1.0
- NP Forum – Software API Conventions Implementation Agreement Revision 1.0
- NP Forum – Software API Framework Implementation Agreement Revision 1.0
- NP Forum – Interface Management API Implementation Agreement Revision 1.0
- NP Forum – IPv4 Unicast Forwarding API (latest draft)
- NP Forum – IPv6 Forwarding API (latest draft)
- NP Forum – MPLS Service API (latest draft)
- RFC 2475, RFC 2597, RFC 2598, RFC 2697, RFC 2698, RFC 3298

Part 2: DiffServ Service Interface Management APIs

2 DiffServ Service Interface Management APIs

2.1 DPE Create

Syntax

```
NPF_error_t NPF_DS_DpeCreate(
            NPF_IN    NPF_callbackHandle_t      cbhandle,
            NPF_IN    NPF_correlator_t          correlator,
            NPF_IN    NPF_errorReporting_t      errorReporting,
            NPF_IN    NPF_IfHandle_t            interface,
            NPF_IN    NPF_uint32_t              noOfEntries,
            NPF_IN    NPF_DS_Dpe_t              *dpe);
```

Description

This function creates one or more DPEs. The callback function receives as many handles as the `NPF_DS_DpeCreate()` can successfully create, and error codes for the rest.

Input Parameters

`cbHandle`	Registered callback handle
`correlator`	Application's context for this call
`errorReporting`	Desired callback
`interface`	Interface to which this DPE is applicable
`noOfEntries`	Number of DPEs to be created
`dpe`	DPE pointer, in which information of the logical DiffServ data path element is stored

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`
`NPF_DS_E_INVALID_DPE`
`NPF_DS_E_INVALID_DPE_PARAM_FILTER`
`NPF_DS_E_INVALID_DPE_PARAM_METER`
`NPF_DS_E_INVALID_DPE_PARAM_DROPPER`
`NPF_DS_E_INVALID_INTERFACE`
`NPF_DS_E_NOT_SUPPORTED`
`NPF_E_UNKNOWN`

Asynchronous Response

A total of `noOfEntries` asynchronous responses (`NPF_DS_AsyncResponse_t`) are passed to the callback function, in one or more invocations. Each response contains one or more DPE handles that identify the unique DPE, or the possible error code.

2.2 DPE Add Association

Syntax

```
NPF_error_t NPF_DS_DpeAddAssociation(
            NPF_IN     NPF_callbackHandle_t    cbhandle,
            NPF_IN     NPF_correlator_t        correlator,
            NPF_IN     NPF_errorReporting_t    errorReporting,
            NPF_IN     NPF_uint32_t            noOfEntries,
            NPF_IN     NPF_DS_DpeSetHandle_t*  dpeSetHandle);
```

Description

This function adds a DPE with `dpeHandle` in the data path following the DPE with `prevDpeHandle`. Packets processed by the DPE with `prevDpeHandle` flow to the DPE with `dpeHandle`.

If the `dpeHandle` handle is the first element in the data flow, the `prevDpeHandle` should be equal to `dpeHandle`.

Input Parameters

`cbHandle`	Registered callback handle
`correlator`	Application's context for this call
`errorReporting`	Desired callback
`noOfEntries`	Number of DPE associations to be made
`dpeSetHandle`	Pair of handles that must be associated

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`
`NPF_DS_E_INVALID_DPE`
`NPF_DS_E_NOT_SUPPORTED`
`NPF_E_UNKNOWN`

Asynchronous Response

The asynchronous response informs whether the DPEs are being added successfully in the data path or not.

2.3 DPE Del Association

Syntax

```
NPF_error_t NPF_DS_DpeDelAssociation(
            NPF_IN    NPF_callbackHandle_t     cbhandle,
            NPF_IN    NPF_correlator_t         correlator,
            NPF_IN    NPF_errorReporting_t     errorReporting,
            NPF_IN    NPF_uint32_t             noOfEntries,
            NPF_IN    NPF_DS_DpeSetHandle_t*   dpeSetHandle);
```

Description

This function disables the association between DPE with handle `dpeHandle` in the data path following the DPE with handle `prevDpeHandle`.

Input Parameters

`cbHandle`	Registered callback handle
`correlator`	Application's context for this call
`errorReporting`	Desired callback
`noOfEntries`	Number of DPE associations that need to be deleted
`dpeSetHandle`	Pair of handles, whose associations are to be broken

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`
`NPF_DS_E_INVALID_DPE`
`NPF_DS_E_NOT_SUPPORTED`
`NPF_E_UNKNOWN`

Asynchronous Response

The asynchronous response informs whether the DPE associations are deleted successfully in the data path or not.

Part 3: DiffServ Service Management APIs

3 DiffServ Service Management APIs

3.1 DPE Modify

Syntax

```
NPF_error_t NPF_DS_DpeModify(
            NPF_IN   NPF_callbackHandle_t    cbhandle,
            NPF_IN   NPF_correlator_t        correlator,
            NPF_IN   NPF_errorReporting_t    errorReporting,
            NPF_IN   NPF_uint32_t            noOfEntries,
            NPF_IN   NPF_DS_DpeHandle_t*     dpeHandle,
            NPF_IN   NPF_DS_Dpe_t*           dpe);
```

Description

This function modifies the existing contents of the DPE in the data processing path. This is an optional function.

Input Parameters

`cbHandle`	Registered callback handle
`correlator`	Application's context for this call
`errorReporting`	Desired callback
`noOfEntries`	Number of DPEs whose parameters are to be modified
`dpeHandle`	DPE handle which is a part of the data flow
`dpe`	DPE pointer in which the information of the logical DiffServ data path element is stored

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`
`NPF_DS_E_INVALID_DPE`
`NPF_DS_E_INVALID_DPE_PARAM_FILTER`
`NPF_DS_E_INVALID_DPE_PARAM_METER`
`NPF_DS_E_INVALID_DPE_PARAM_DROPPER`
`NPF_DS_E_INVALID_DPE_HANDLE`
`NPF_DS_E_NOT_SUPPORTED`
`NPF_E_UNKNOWN`

Asynchronous Response

The asynchronous response contains the possible error code.

3.2 DPE Delete

Syntax

```
NPF_error_t NPF_DS_DpeDelete(
            NPF_IN   NPF_callbackHandle_t   cbhandle,
            NPF_IN   NPF_correlator_t       correlator,
            NPF_IN   NPF_errorReporting_t   errorReporting,
            NPF_IN   NPF_uint32_t           nHandles,
            NPF_IN   NPF_DS_DpeHandle_t*    dpeHandle);
```

Description

This function deletes one or more DPEs from the data processing path.

Input Parameters

`cbHandle`	Registered callback handle
`correlator`	Application's context for this call
`errorReporting`	Desired callback
`nHandles`	Number of DPEs to delete
`dpeHandle`	Array of DPE handles which is a part of the data flow

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`
`NPF_DS_E_INVALID_DPE_HANDLE`
`NPF_DS_E_NOT_SUPPORTED`
`NPF_E_UNKNOWN`

Asynchronous Response

The response contains the possible error code.

3.3 DPE Query Attributes

Syntax

```
NPF_error_t NPF_DS_DpeQueryAttributes(
            NPF_IN   NPF_callbackHandle_t   cbhandle,
            NPF_IN   NPF_correlator_t       correlator,
            NPF_IN   NPF_errorReporting_t   errorReporting,
            NPF_IN   NPF_uint32_t           nHandles,
            NPF_IN   NPF_DS_DpeHandle_t*    dpeHandle
);
```

Description

This function queries the attributes of the DPE by passing its handle. This function returns the contents of the DPE pointed by the handle. This is an optional function.

Input Parameters

`cbHandle`	Registered callback handle
`correlator`	Application's context for this call
`errorReporting`	Desired callback
`nHandles`	Number of DPEs whose attributes must be queried
`dpeHandle`	Array of DPE handles which is part of a data flow

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`
`NPF_DS_E_INVALID_DPE_HANDLE`
`NPF_DS_E_NOT_SUPPORTED`
`NPF_E_UNKNOWN`

Asynchronous Response

The asynchronous response contains either the structure of a DPE, or the possible error code.

3.4 DPE Query Statistics

Syntax

```
NPF_error_t NPF_DS_DpeQueryStats(
            NPF_IN   NPF_callbackHandle_t    cbhandle,
            NPF_IN   NPF_correlator_t        correlator,
            NPF_IN   NPF_errorReporting_t    errorReporting,
            NPF_IN   NPF_uint32_t            nHandles,
            NPF_IN   NPF_DS_DpeHandle_t*     dpeHandle
);
```

Description

This function queries the statistics of the DPE by passing its handle, and returns the statistics of the DPE pointed by the handle. This is an optional function.

Input Parameters

`cbHandle`	Registered callback handle
`correlator`	Application's context for this call
`errorReporting`	Desired callback
`nHandles`	Number of DPEs whose statistics must be given
`dpeHandle`	Array of DPE handles which is part of a data flow

Output Parameters

None.

Asynchronous Error Codes

`NPF_NO_ERROR`
`NPF_DS_E_INVALID_DPE_HANDLE`
`NPF_DS_E_NOT_SUPPORTED`
`NPF_E_UNKNOWN`

Asynchronous Response

The asynchronous response contains either the statistics structure of the DPE, or the possible error code.

Part 4: DPE Functions

4 DPE Functions

This section describes the functions of DiffServ Service APIs.

4.1 Completion Callback Function

Syntax

```
typedef void (*NPF_DS_CallbackFunc_t) (
               NPF_IN   NPF_userContext_t      context,
               NPF_IN   NPF_correlator_t       correlator,
               NPF_IN   NPF_DS_CallbackData_t  *callbackData);
```

Description

The application registers this asynchronous response handling routine to the API implementation. The callback function is implemented by the application, and is registered to the API implementation through the `NPF_DS_Register()` function.

Input Parameters

`userContext`	The context item that is supplied by the application during the registration of completion callback function
`correlator`	The correlator item or call ID that is supplied by the application when an API function call is made. The correlator is used by the application to distinguish between multiple invocations of the same function.
`CallbackData:`	Pointer to a structure containing an array of response information related to the API function call. It contains information that is common to all functions and specific to a particular function. Refer the `NPF_DS_CallbackData_t` definition for details.

Output Parameters

None.

Error Codes

None.

4.2 Completion Callback Registration Function

Syntax

```
NPF_error_t NPF_DS_Register (
            NPF_IN     NPF_userContext_t       context,
            NPF_IN     NPF_DS_CallbackFunc_t   callback,
            NPF_OUT    NPF_callbackHandle_t    *cbhandle);
```

Description

This function is used by an application to register its completion callback function for receiving the asynchronous responses related to API function calls. The application can register multiple callback functions using this function.

The callback function is identified by the `userContext` and `callback` pair, and for each individual pair, a unique `cbHandle` is assigned for future reference. Duplicate registration of the same callback function with different `userContext` is allowed, since the callback function is identified by both `userContext` and `callback`.

In addition, the same `userContext` can be shared among different callback functions. Duplicate registration of the same `userContext` and callback pair has no effect. It outputs a handle that is already assigned to the pair, and returns `NPF_E_ALREADY_REGISTERED`.

Note: `NPF_DS_Register()` is a synchronous function and has no completion callback associated with it.

Input Parameters

`userContext`	Context item to identify unique context of the application, registering the completion callback function. When it is called, the exact value is provided back to the registered completion callback function as its first parameter. The application can assign any value to the `userContext` and the value is completely opaque to the API implementation.
`Callback`	Pointer to the completion callback function to be registered

Output Parameters

`cbHandle`	Unique ID assigned for the registered `userContext` and `callback` pair. This handle is used by the application to specify the callback to be called when invoking asynchronous API functions. It is also used during the de-registering of the `userContext` and `callback` pair.

Error Codes

`NPF_NO_ERROR`	Registration successfully completed
`NPF_E_BAD_CALLBACK_FUNCTION`	`callbackFunc` is `NULL`
`NPF_E_ALREADY_REGISTERED`	Function already registered

4.3 Completion Callback De-registration Function

Syntax

```
NPF_error_t NPF_DS_Deregister(
            NPF_IN   NPF_callbackHandle_t   cbhandle);
```

Description

This function is used by the application to de-register a `userContext` and `callback` function pair. No more function calls can be made using the de-registered callback handle on return of the de-register function.

Input Parameters

`cbHandle`	Unique ID, representing the `user context` and `callback` function pair to be de-registered

Return Values

`NPF_NO_ERROR`	De-registration successfully completed
`NPF_E_BAD_CALLBACK_HANDLE`	The API implementation does not recognize the callback handle. There is no effect to the registered callback functions.

Part 5: Data Structures

5 Data Structures

This section describes DiffServ API data structure definitions. Figure 2 explains the relationships between the DPEs defined in this document. Some DPE elements can be present or absent for a particular flow depending on the location, which can be on the ingress or egress side of the system.

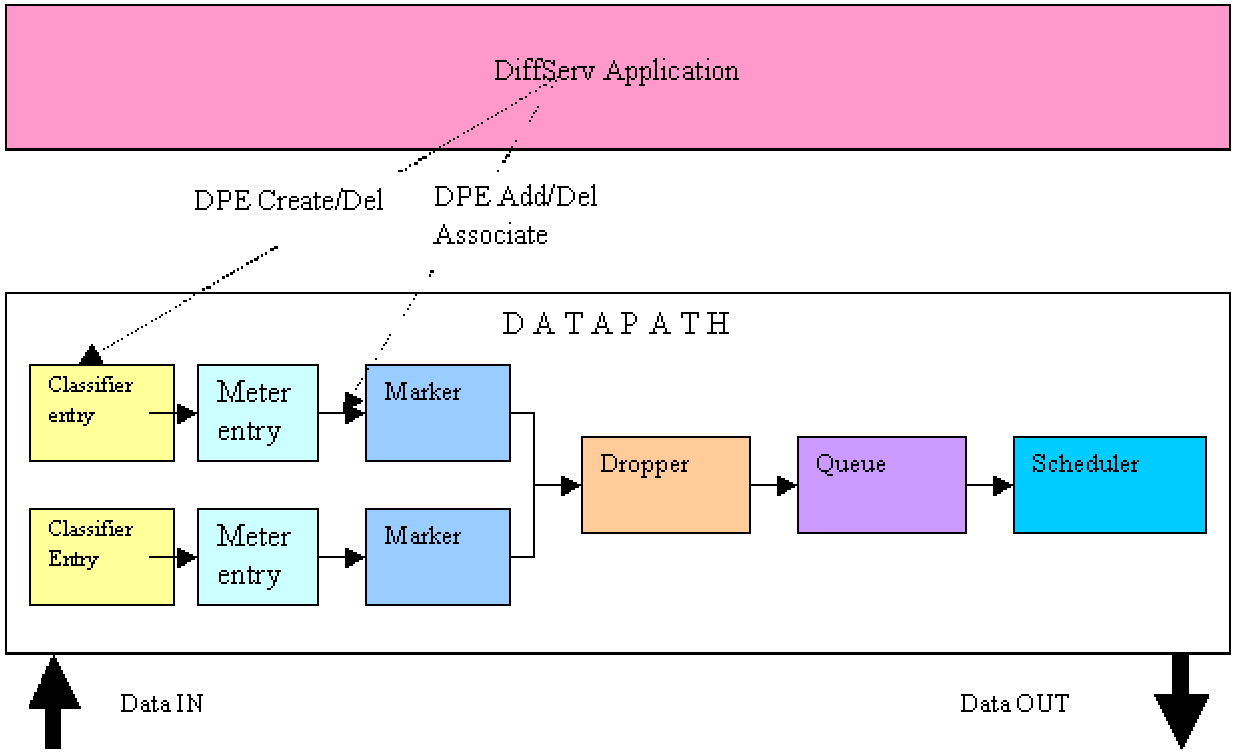

Figure 2 High-level overview of data path element

In the ingress classification, DPE followed by the meter and the marker DPE can be queued and scheduled to reach the egress side of the system. In the dropper followed by queue, the scheduler and shaper can be present at the egress.

5.1 NPF DiffServ DPE Data Types

5.1.1 NPF_DS_DpeHandle_t – DPE handle

```
typedef     NPF_uint32_t      NPF_DS_DpeHandle_t;
```

5.1.2 NPF_DS_DpeType_t – DPE Type

This enumerated type definition is used to indicate the DPE to be used.

```
typedef enum
{
    NPF_DS_CLASSIFIER = 1,
    NPF_DS_METER = 2,
    NPF_DS_MARKER = 3,
    NPF_DS_DROPPER = 4,
    NPF_DS_QUEUE = 5,
    NPF_DS_SCHEDULER = 6,
    NPF_DS_SHAPER = 7
}NPF_DS_DpeType_t;
```

5.1.3 NPF_DS_IPv4Filter_t – IPV4 Filter

This structure defines the classification characteristics for IPV4 data.

```
typedef struct
{
  NPF_IPv4Address_t dstAddr;
  /* IPv4 destination address */


  NPF_uint8_t       dstNetpLen
  /* IPv4 destination Prefix length in bits (1-32) */


  NPF_IPv4Address_t srcAddr;
  /* IPv4 source address */


  NPF_uint8_t       srcNetpLen
  /* IPv4 source Prefix length in bits (1-32) */


  NPF_int32_t       dscp;
  /* DiffServ codepoint stored in Type of Service field of an IPv4 */
  /* header */


  NPF_int32_t       protocol;
  /* Layer 4 protocol number (e.g. TCP /* or UDP) */


  NPF_uint32_t      dstL4Port;
  /* Destination port number of a layer 4 protocol (e.g. UDP or */
  /* TCP)*/


  NPF_uint32_t      srcL4Port;
  /* Source port number of a layer 4 protocol (e.g. UDP or TCP)*/


}NPF_DS_IPv4Filter_t;
```

5.1.4 NPF_DS_IPv6Filter_t – IPV6 Filter

This structure defines the classification characteristics for IPV6 data.

```
typedef struct
{
 NPF_IPv6Address_t   dstAddr;
 /* IPv6 destination address */


 NPF_uint8_t         dstNetpLen
 /* IPv6 destination Prefix length in bits (1-128) */


 NPF_IPv6Address_t   srcAddr;
 /* IPv6 source address */


 NPF_uint8_t         srcNetpLen
 /* IPv6 source Prefix length in bits (1-128) */


 NPF_int32_t         fid;
 /* Flow label in the IPV6 header */


 NPF_int32_t         dscp;
 /* DiffServ codepoint stored in Type of Service field of an IPv6 */
 /* header */


 NPF_int32_t         protocol;
 /* Layer 4 protocol number (e.g. TCP or UDP) */


 NPF_uint32_t        dstL4Port;
 /* Destination port number of a layer 4 protocol (e.g. UDP or TCP)*/


 NPF_uint32_t        srcL4Port;
 /* Source port number of a layer 4 protocol (e.g. UDP or TCP)*/


}NPF_DS_IPv6Filter_t;
```

5.1.5 NPF_DS_MPLS_FilterType_t – MPLS Filter Type

This enumerated type definition is used to indicate the filter type to be used for MPLS filter.

```
typedef enum
{
    NPF_DS_MPLS_FEC = 1,
    NPF_DS_MPLS_LABEL = 2
}NPF_DS_MPLS_FilterType_t;
```

5.1.6 NPF_MPLS_Filter_t – MPLS Filter

This structure gives the MPLS filter parameters.

```
typedef struct
{
 NPF_DS_MPLS_FilterType_t  type;   /* MPLS Filter Type */
  union
     {
       NPF_MPLS_Fec_t      fec;
       NPF_MPLS_Label_t    label;
     } filter;
}NPF_DS_MPLS_Filter_t;
```

Note: Please refer the MPLS Service API for the structure Fec and label.

5.1.7 NPF_DS_ClassifierType_t – Classifier DPE Type

This enumerated type definition is used to indicate supported classification types.

```
typedef enum
{
    NPF_DS_IPV4_FILTER = 1,
    NPF_DS_IPV6_FILTER = 2,
    NPF_DS_MPLS_FILTER = 3
}NPF_DS_ClassifierType_t;
```

5.1.8 NPF_DS_Filter_t – Filter

This structure is the union of a three-filter structure.

```
typedef union
    {
      NPF_DS_IPV4_Filter_t    filter_IPV4;      /* IPV4 Filter */
      NPF_DS_IPV6_Filter_t    filter_IPV6;      /* IPV6 Filter */
      NPF_DS_MPLS_Filter_t    filter_MPLS;      /* MPLS Filter */
    }NPF_DS_Filter_t;
```

5.1.9 NPF_DS_Classifier_t – Classifier DPE Entry

This structure type definition is used to store one classification entry or an array of classification entries.

```
typedef struct
{
   NPF_DS_ClassifierType_t      type;
   /* Classifier type */


   NPF_uint32_t                 precedence;
   /* ColorId, Packet drop precedence level: green, yellow or red */
   /* color */


   NPF_DS_Filter_t              filterEntry;
   /* filter Entry */


}NPF_DS_Classifier_t;
```

5.1.10 NPF_DS_MeterType_t – Meter Type

This enumerated type definition is used to indicate the various meter types that are supported.

```
typedef enum {
  NPF_DS_SIMPLE_TB = 1,
  /* Simple Token Bucket Metering */


  NPF_DS_AVG_RATE = 2,
  /* Average rate metering */


  NPF_DS_SRTCM_BLIND = 3,
  /*Single Rate Tricolor metering in color blind  */


  NPF_DS_SRTCM_AWARE = 4,
  /* SRTCM color aware mode */


  NPF_DS_TRTCM_BLIND = 5,
  /* Two Rate Tricolor Metering in color blind */


  NPF_DS_TRTCM_AWARE = 6,
  /* TRTCM color aware mode */


  NPF_DS_TSW_TCM = 7
  /*Time Sliding Window TCM */


} NPF_DS_MeterType_t;
```

5.1.11 NPF_DS_MarkerType_t – Marker Type

This enumerated type definition is used to indicate the marker types that are supported.

```
 typedef enum {
    NPF_DS_MARKER_DSCP = 1,
    NPF_DS_MARKER_MPLS = 2
} NPF_DS_MarkerType_t;
```

5.1.12 NPF_DS_DSCP_Marker_t – DSCP Marker

This structure contains the `dscp` values that are to be marked in the data header.

```
typedef struct
{
  NPF_uint8_t greenDscp;  /* DSCP value to mark for green packets */
  NPF_uint8_t yellowDscp; /* DSCP value to mark for yellow packets */
  NPF_uint8_t redDscp;    /* DSCP value to mark for red packets */
}NPF_DS_DSCP_Marker_t;
```

5.1.13 NPF_DS_MPLS_Marker_t – MPLS Marker

This structure contains the label values that are to be marked in the data header.

```
typedef struct
{
   NPF_MplsLabel_t  tos_label;
   /*top of stack label including Exp Bits */
}NPF_DS_MPLS_Marker_t;
```

5.1.14 NPF_DS_MarkerEntry_t – Marker DPE Entry

This structure contains the marker entry.

```
typedef union
{
      NPF_DS_DSCP_Marker_t    dscp;
      NPF_DS_MPLS_Marker_t    label;
}NPF_DS_MarkerEntry_t;
```

5.1.15 NPF_DS_Marker_t – Marker DPE

This structure contains the marker DPE.

```
typedef struct
{
   NPF_DS_MarkerType_t  type;
```

```
   NPF_boolean_t        statsRequired;
   /* boolean value; if true, statistics are gathered */


   NPF_DS_MarkerEntry_t markerEntry;
   /* Marker Entry */


}NPF_DS_Marker_t;
```

5.1.16 NPF_DS_MeterEntry_t – Meter DPE Entry

This structure contains the meter entry in the meter DPE.

```
typedef struct
{
   NPF_uint32_t      cir;
   /* Committed Information Rate in kbits/sec */


   NPF_uint32_t      pir;
   /* Peak Information Rate in kbits/sec, used in */
   /* NPF_DS_TRTCM_AWARE type */


   NPF_uint32_t      cbs;
   /* Committed Burst Size in bytes */


   NPF_uint32_t      pbs;
   /* Peak Burst Size in bytes, used in NPF_DS_TRTCM_AWARE type */


   NPF_uint32_t      ebs;
   /* Excess Burst Size in bytes */


   NPF_DS_DpeType_t  nextGreenDPE;
   /* The next DPE should be attached for conformance (Green) flow */


   NPF_DS_DpeType_t  nextYellowDPE;
   /* The next DPE should be attached for */
   /* non-conformance-tolerable Yellow flow */


   NPF_DS_DpeType_t  nextRedDPE;
   /* The next DPE should be attached for non conformance */
   /* (Red) flow */


}NPF_DS_MeterEntry_t;
```

5.1.17 NPF_DS_Meter_t – Meter DPE

This structure contains the meter DPE.

```
typedef struct
{
   NPF_DS_MeterType_t   type;

   NPF_boolean_t        statsRequired;
   /* boolean value; if true, statistics are gathered */

   NPF_DS_MeterEntry_t  meterEntry;
   /* one Meter Entry */

}NPF_DS_Meter_t;
```

5.1.18 NPF_DS_DropperType_t – Dropper Type

This enumerated type definition is used to indicate the dropper types that are supported.

```
typedef enum
{
    NPF_DS_RED = 1,     /* Random Early Detection */
    NPF_DS_WRED = 2     /* Weighted Random Early Detection */
}NPF_DS_DropperType_t;
```

5.1.19 NPF_DS_RED_Instance_t – RED Parameter

This structure contains the required RED parameters.

```
typedef struct
{
   NPF_uint8_t   minThreshold;
   /* minimum queue threshold in packets (0-255) */
   /* no packets are dropped, if the average queue length is */
   /* below this threshold */

   NPF_uint8_t   maxThreshold;
   /* maximum queue threshold in packets (0-255) */
   /* packets are always dropped, if the average queue length */
   /* exceeds this threshold */

   NPF_uint16_t  maxProbability;
   /* dropping probability at maxThreshold, expressed as a */
   /* fraction with denominator of 65535 */

}NPF_DS_RED_Instance_t;
```

5.1.20 NPF_DS_DropperEntry_t – Dropper DPE Entry

This structure contains the dropper entry.

```
typedef struct
{
 NPF_DS_RED_Instance_t   green;   /* RED params for green packets */
 NPF_DS_RED_Instance_t   yellow;  /* RED params for yellow packets */
 NPF_DS_RED_Instance_t   red;     /* RED params for red packets */
}NPF_DS_DropperEntry_t;
```

5.1.21 NPF_DS_Dropper_t – Dropper DPE

This structure contains the dropper DPE.

```
typedef struct
{
 NPF_DS_DropperType_t      type;
 /*Dropper Alogorithm */


 NPF_boolean_t             statsRequired;
 /* boolean value; if true, statistics are gathered */


 NPF_uint32_t              serviceRate;
 /* service rate in packets/sec */


 NPF_uint8_t               weight;
 /* used in WRED */


 NPF_DS_DropperEntry_t     dropperEntry;
 /* one dropper Entry */


}NPF_DS_Dropper_t;
```

5.1.22 NPF_DS_QueueType_t – Queue Type

This enumerated type definition is used to indicate the various queues that are supported.

```
typedef enum
{
    NPF_DS_SPQ =  1,    /* Strict Priority Queuing */
    NPF_DS_CBQ =  2,    /* Class Based Queuing */
    NPF_DS_CBWFQ =3     /* Class Based Weighted Fair Queuing */
}NPF_DS_QueueType_t;
```

5.1.23 NPF_DS_ClassType_t – Queue Class Type

This enumerated type definition is used to indicate the various classes that the queues support.

```
typedef enum
{
    NPF_DS_CLASS_EF =  1,          /* EF class */
    NPF_DS_CLASS_AF1 =  2,         /* AF class-1 */
    NPF_DS_CLASS_AF2 =  3,         /* AF class-2 */
    NPF_DS_CLASS_AF3 =  4,         /* AF class-3 */
    NPF_DS_CLASS_AF4 =  5,         /* AF class-4 */
    NPF_DS_CLASS_BF =   6          /* BF class */
}NPF_DS_ClassType_t;
```

5.1.24 NPF_DS_MinRate_t – Minimum Rate

This structure contains the minimum bandwidth or baud rate for the queue or scheduler.

```
typedef struct
{
NPF_uint32_t      priority;
/* Priority of input to the associated Queue/scheduler */


NPF_uint32_t      abs;
/* Minimum absolute rate, in kbps that downstream queue/scheduler */
/* should allocate*/


NPF_uint32_t      rel;
/* Minimum rate that downstream scheduler should allocate to this */
/* queue */


} NPF_DS_MinRate_t;
```

5.1.25 NPF_DS_SlMaxRate_t – Maximum Rate

This structure contains the single-rate maximum bandwidth or baud rate for the queue or scheduler.

```
typedef struct
{
 NPF_uint32_t    level;
 /*indicates which level of multi rate shaper being given */
 /* its parameters */


 NPF_uint32_t    abs;
 /*Maximum rate, in kbps that downstream scheduler should allocate */
 /* to this queue */


 NPF_uint32_t    rel;
 /*The maximum rate that a downstream  scheduler element should */
 /* allocate to this queue, relative to the maximum rate of the */
 /* interface as reported by ifSpeed in units of 1/1000 of 1.*/


 NPF_uint32_t    threshold;
 /*The number of bytes of queue depth at which the rate of a */
 /* multi-rate scheduler will increase to next output rate */


} NPF_DS_SlMaxRate_t;
```

5.1.26 NPF_DS_MlMaxRate_t – Maximum Rate

This structure contains the multi-rate maximum bandwidth or baud rate for the queue or scheduler.

```
typedef struct
{
 NPF_uint32_t       maxRateCount;  /* No of multi-rate entries */
 NPF_DS_SlMaxRate_t *maxRateList;  /* Array of multirate entries */
} NPF_DS_MlMaxRate_t;
```

5.1.27 NPF_DS_MaxRateType_t – Maximum Rate Type

This enumerated type definition is used to indicate the usability of single or multiple max rates.

```
typedef enum {
    NPF_DS_SL_MAXRATE = 1,    /* Single rate */
    NPF_DS_ML_MAXRATE = 2     /* Multi rate */
} NPF_DS_MaxRateType_t;
```

5.1.28 NPF_DS_MaxRate_t – Maximum Rate

This structure contains the maximum bandwidth or baud rate for the queue or scheduler.

```
typedef struct
{
    NPF_DS_MaxRateType_t        type;
    union
    {
        NPF_DS_SlMaxRate_t      slMaxRate;
        NPF_DS_MlMaxRate_t      mlMaxRate;
    } maxRate;
} NPF_DS_MaxRate_t;
```

5.1.29 NPF_DS_Q_t – Queue DPE

This structure represents the queue DPE.

```
typedef struct
{
    NPF_DS_QueueType_t        type;
    NPF_DS_ClassType_t        classId;
    NPF_DS_MinRate_t          minRate;
    NPF_DS_MaxRate_t          maxRate;
} NPF_DS_Q_t;
```

5.1.30 NPF_DS_SchedulerType_t – Scheduler Type

This enumerated type definition is used to indicate the scheduler types that are supported.

```
typedef enum
{
  NPF_DS_SCHEDULER_SP = 1,     /* Strict Priority Scheduler */
  NPF_DS_SCHEDULER_RR = 2,     /* Round Robin Scheduler*/
  NPF_DS_SCHEDULER_WRR = 3,    /* Weighted Round Robin Scheduler*/
  NPF_DS_SCHEDULER_WFQ = 3,    /* Weighted Fair Queuing scheduler*/
  NPF_DS_SCHEDULER_DRR = 4     /* Deficit Round Robin Scheduler*/
}NPF_DS_SchedMethod_t;
```

5.1.31 NPF_DS_Scheduler_t – Scheduler DPE

This structure represents the scheduler DPE.

```
typedef struct
{
    NPF_DS_SchedMethod_t      type;
    NPF_DS_MinRate_t          minRate;
    NPF_DS_MaxRate_t          maxRate;
} NPF_DS_Scheduler_t;
```

5.1.32 NPF_DS_Dpe_t – DPE

This structure represents the DPE.

```
typedef struct
{
    NPF_DS_DpeType_t          type;
    union
    {
      NPF_DS_Classifier_t     classifier;
      NPF_DS_Marker_t         marker;
      NPF_DS_Meter_t          meter;
      NPF_DS_Dropper_t        dropper;
      NPF_DS_Q_t              queue;
      NPF_DS_Scheduler_t      scheduler;
    }u;
}NPF_DS_Dpe_t;
```

5.1.33 NPF_DS_DpeSetHandle_t – DPE Handle Pair

This structure contains a pair of DPE handles that can be associated.

```
typedef struct
{
NPF_DS_DpeHandle_t  dpeHandle;     /* handle of DPE part of a flow */
NPF_DS_DpeHandle_t  prevDpeHandle; /* handle of DPE part of a flow */
} NPF_DS_DpeSetHandle_t;
```

5.2 NPF DPE Statistics Data Types

5.2.1 NPF_DS_ClassifierStats_t

This structure type definition holds the statistics associated with the classifier entry.

```
typedef struct
{
  NPF_DS_Counter_t       defaultPackects;
  NPF_DS_Counter_t       defaultBytes;
  NPF_DS_Counter_t       totalBytes;
  NPF_DS_Counter_t       totalPackets;
}NPF_DS_ClassifierStats_t;
```

5.2.2 NPF_DS_ColorAwareMeterStats_t

This structure type definition holds the statistics associated with the color aware meter entry.

```
typedef struct {
  NPF_DS_Counter_t       greenByteCount;
  NPF_DS_Counter_t       greenPacketCount;
  NPF_DS_Counter_t       yellowByteCount;
  NPF_DS_Counter_t       yellowPacketCount;
  NPF_DS_Counter_t       redByteCount;
  NPF_DS_Counter_t       redPacketCount;
} NPF_DS_ColorAwareMeterStats_t;
```

5.2.3 NPF_DS_SimpleMeterStats_t

This structure type definition holds the statistics associated with the simple meter entry.

```
typedef struct {
  NPF_DS_Counter_t       confByteCount;
  NPF_DS_Counter_t       confPacketCount;
  NPF_DS_Counter_t       nonConfByteCount;
  NPF_DS_Counter_t       nonConfPacketCount;
} NPF_DS_SimpleMeterStats_t;
```

5.2.4 NPF_DS_MeterStats_t

This structure type definition holds the statistics associated with the meter entry.

```
typedef struct
{
 NPF_DS_MeterType_t               type;   /* Meter type */
 union
 {
  NPF_DS_ColorAwareMeterStats_t   colorAwareStats;
  /*meant for all color aware metering  algorithms */

  NPF_DS_SimpleMeterStats_t       simpleStats;
 }u;
}NPF_DS_MeterStats_t;
```

5.2.5 NPF_DS_MarkerStats_t

This structure type definition holds the statistics associated with the marker entry.

```
typedef struct
{
 NPF_DS_Counter_t       bytesMarked;
 NPF_DS_Counter_t       packetsMarked;
}NPF_DS_MarkerStats_t;
```

5.2.6 NPF_DS_ColorAwareDropperStats_t

This structure type definition holds the statistics associated with the color aware dropper entry.

```
typedef struct
{
 NPF_DS_Counter_t  greenByteDropped;      /* Green bytes dropped */
 NPF_DS_Counter_t  greenPacketDropped;  /* Green packets dropped */
 NPF_DS_Counter_t  yellowByteDropped;   /* Yellow bytes dropped */
 NPF_DS_Counter_t  yellowPacketDropped; /* Yellow packets dropped */
 NPF_DS_Counter_t  redByteDropped;      /* Red bytes dropped */
 NPF_DS_Counter_t  redPacketDropped;    /* Red packets dropped */
}NPF_DS_ColorAwareDropperStats_t;
```

5.2.7 NPF_DS_SimpleDropperStats_t

This structure type definition holds the statistics associated with the simple dropper entry.

```
typedef struct
{
  NPF_DS_Counter_t  bytesConsumed;
  NPF_DS_Counter_t  packetsConsumed;
  NPF_DS_Counter_t  bytesDropped;
  NPF_DS_Counter_t  packetsDropped;
}NPF_DS_SimpleDropperStats_t;
```

5.2.8 NPF_DS_DropperStats_t

This structure type definition holds the statistics associated with the dropper entry.

```
typedef struct
{
    NPF_DS_MeterType_t        type;             /* Meter type */
    union
    {
      NPF_DS_ColorAwareDropperStats_t     colorAwareDropperStats;
      NPF_DS_SimpleDropperStats_t         simpleDropperStats;
    }u;
}NPF_DS_DropperStats_t;
```

5.3 Data Structures for Completion Callbacks

This section describes completion callback functions, and their associated data structures.

```
/*
 *    DiffServ DPE Completion Callback Types
 */

typedef enum {
  NPF_DS_DPE_CREATE = 1,
  NPF_DS_DPE_ADD = 2,
  NPF_DS_DPE_MODIFY = 3,
  NPF_DS_DPE_DELETE = 4,
  NPF_DS_DPE_QUERY = 5,
  NPF_DS_DPE_STATISTICS = 6
} NPF_DS_CallbackType_t;

typedef struct
{
```

```
  union
  {
    NPF_DS_ClassifierStats_t   classfierStats;
    NPF_DS_MeterStats_t        meterStats;
    NPF_DS_MarkerStats_t    markerStats;
    NPF_DS_DropperStats_t   dropperStats;
    NPF_DS_QueueStats_t     queueStats;
    NPF_DS_SchedulerStats_t schedulerStats;
  } u;
} NPF_DS_StatsResp_t;

typedef struct
{
  NPF_DS_ErrorType_t     error;
  NPF_DS_DPE_Type_t      type;
  union
  {
    NPF_DS_DpeHandle_t   handle;
    NPF_DS_Dpe_t         dpe;
    NPF_DS_StatsResp_t   statsResp;
  } u;
} NPF_DS_AsyncResponse_t;

typedef struct
{
  NPF_DS_CallbackType_t  type;
  NPF_boolean_t          allOK;
  NPF_uint32_t           nResp;
  NPF_DS_AsyncResponse_t *resp;
} NPF_DS_CallbackData_t;

typedef void (*NPF_DS_CallbackFunc_t) (
  NPF_IN   NPF_userContext_t         context,
  NPF_IN   NPF_correlator_t          correlator,
  NPF_IN   NPF_DS_CallbackData_t     *callbackData);
```