[UEFI Basics] EDK Network Framework (UDP4)

原创 jiangwei0512 Posted on 2024-01-21 08:04:33 Read 1.5k Collection 24 Likes 23

Category Column: UEFI Development Basics Article Tags: network uefi

摘要 This article focuses on the UDP4 protocol, and introduces its characteristics of not providing complex control mechanisms and using IP for connectionless communication. It also explains the UDP4 code implementation, including entry and port initialization. It also explains in detail the implementation of related structures such as UDP4_SERVICE_DATA, UDP4_INSTANCE_DATA, etc., as well as the implementation of each function of EFI_UDP4_PROTOCOL, and gives code examples.

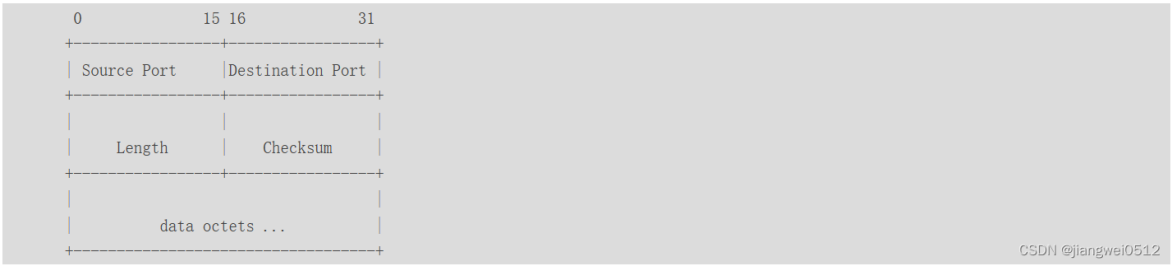
The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

UDP4

UDP4 Protocol Description

UDP stands for **User Datagram** Protocol . It does not provide a complex control mechanism, but only uses IP **to** provide connectionless communication services. It sends the data sent by the upper-layer application to the network as it is when it is received.

UDP message format :



The parameters are described as follows:

Fields	Length (bytes)	describe
Source Port	2	Send port, identifies which application sends (sending process).
Destination Port	2	Destination port, which identifies which application receives (receiving process).
Length	2	The minimum number of bytes of the UDP header plus the UDP data is 8.
Checksum	2	Overrides the UDP header and UDP data, which is optional.
data octets	Lengthening	UDP payload, optional.

The first four parameters correspond to the UDP header in the UEFI code:

c AI generated projects 登录复制 run

```c
//
// UDP header definition
//
typedef struct {
  UINT16    SrcPort;
  UINT16    DstPort;
  UINT16    Length;
  UINT16    Checksum;
} EFI_UDP_HEADER;
```

UDP4 code overview

UDP4 is also a common network protocol. In fact, it is now NetworkPkg\Udp4Dxe\Udp4Dxe.inf. Here we first need to look at its entry:

c AI generated projects 登录复制 run

```c
EFI_STATUS
EFIAPI
Udp4DriverEntryPoint (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
  //
  // Install the Udp4DriverBinding and Udp4ComponentName protocols.
  //
  Status = EfiLibInstallDriverBindingComponentName2 (
             ImageHandle,
             SystemTable,
             &gUdp4DriverBinding,
             ImageHandle,
             &gUdp4ComponentName,
             &gUdp4ComponentName2
             );
  if (!EFI_ERROR (Status)) {
    //
    // Initialize the UDP random port.
    //
    mUdp4RandomPort = (UINT16)(((UINT16)NetRandomInitSeed ()) % UDP4_PORT_KNOWN + UDP4_PORT_KNOWN); // 宏的值是1024
  }
}
```

Because UDP4 is also a UEFI Driver Model, the first step is to install it `gUdp4DriverBinding` . The implementation is:

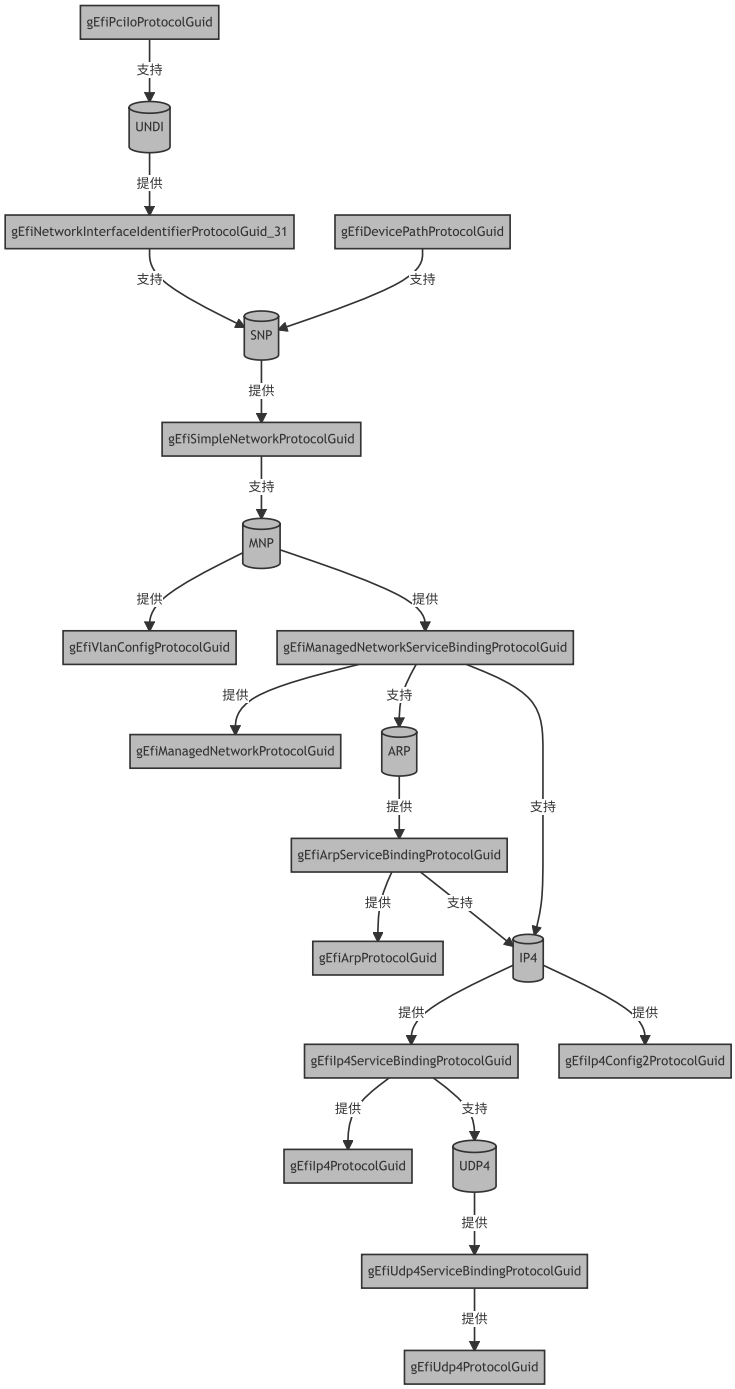
c AI generated projects 登录复制 run

```c
EFI_DRIVER_BINDING_PROTOCOL  gUdp4DriverBinding = {
  Udp4DriverBindingSupported,
  Udp4DriverBindingStart,
  Udp4DriverBindingStop,
  0xa,
  NULL,
```

```
  NULL
};
```

The second step is to initialize a random UDP port. According to the common network protocol, the UDP port occupies two bytes (i.e. 16 bits). It can be any port as long as it is not a recognized port in the range of 0-1023, and it does not matter if it is consistent with the TCP port.

Relationship diagram of UDP4 in UEFI network protocol stack:



Udp4DriverBindingSupported

UDP4 depends on IP4 :

c AI generated projects 登录复制 run

```c
EFI_STATUS
EFIAPI
Udp4DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL  *This,
  IN EFI_HANDLE                   ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL     *RemainingDevicePath  OPTIONAL
  )
{
  //
  // Test for the Ip4 Protocol
  //
  Status = gBS->OpenProtocol (
                ControllerHandle,
                &gEfiIp4ServiceBindingProtocolGuid,
                NULL,
                This->DriverBindingHandle,
                ControllerHandle,
                EFI_OPEN_PROTOCOL_TEST_PROTOCOL
                );
}
```

Udp4DriverBindingStart

The flow of the Start function is as follows:

1. initialization `UDP4_SERVICE_DATA` .
2. Install `gEfiUdp4ServiceBindingProtocolGuid` .

Like other drivers, the focus is also on the structure, which is here `UDP4_SERVICE_DATA` .

UDP4_SERVICE_DATA

`UDP4_SERVICE_DATA` Create in the Start function:

```c
EFI_STATUS
EFIAPI
Udp4DriverBindingStart (
  IN EFI_DRIVER_BINDING_PROTOCOL  *This,
  IN EFI_HANDLE                   ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL     *RemainingDevicePath  OPTIONAL
  )
{
  Status = Udp4CreateService (Udp4Service, This->DriverBindingHandle, ControllerHandle);
}
```

Its structure is defined as follows:

```c
typedef struct _UDP4_SERVICE_DATA_ {
  UINT32                          Signature;
  EFI_SERVICE_BINDING_PROTOCOL    ServiceBinding;
  EFI_HANDLE                      ImageHandle;
  EFI_HANDLE                      ControllerHandle;
  LIST_ENTRY                      ChildrenList;
  UINTN                           ChildrenNumber;
  IP_IO                           *IpIo;

  EFI_EVENT                       TimeoutEvent;
} UDP4_SERVICE_DATA;
```

Compared with the previous service data, this structure is quite simple, and the more important members are:

- `ServiceBinding` :correspond `mUdp4ServiceBinding` :

```c
EFI_SERVICE_BINDING_PROTOCOL  mUdp4ServiceBinding = {
  Udp4ServiceBindingCreateChild,
  Udp4ServiceBindingDestroyChild
};
```

Used to create UDP4 subkey.

- `ChildrenList` , `ChildrenNumber` : The corresponding `UDP4_INSTANCE_DATA` structure, `Udp4ServiceBindingCreateChild()` created by , is a structure representing sub-items, which will be further introduced in UDP4_INSTANCE_DATA .
- `IpIo` ： It is a wrapper for IP4 instance, UDP4 communicates through it, which has been introduced in IP_IO . You will see similar structures in UDP4 and TCP4 `IP_IO` , which are just guarantees for the IP layer. The reason for such wrappers is that there are two versions, IPv4 and IPv6, and here we only focus on IPv4.
- `TimeoutEvent` : A timed event, created in `Udp4CreateService()` the function:

```c
EFI_STATUS
Udp4CreateService (
  IN OUT UDP4_SERVICE_DATA  *Udp4Service,
  IN     EFI_HANDLE         ImageHandle,
  IN     EFI_HANDLE         ControllerHandle
  )
{
  //
  // Create the event for Udp timeout checking.
  //
  Status = gBS->CreateEvent (
                  EVT_TIMER | EVT_NOTIFY_SIGNAL,
                  TPL_CALLBACK,
                  Udp4CheckTimeout,
                  Udp4Service,
                  &Udp4Service->TimeoutEvent
                  );
  //
  // Start the timeout timer event.
  //
  Status = gBS->SetTimer (
                  Udp4Service->TimeoutEvent,
                  TimerPeriodic,
                  UDP4_TIMEOUT_INTERVAL // 50 milliseconds
                  );
```

The corresponding callback function `Udp4CheckTimeout()` is used to detect whether the received message is expired. Its main code is:

```c
VOID
EFIAPI
Udp4CheckTimeout (
  IN EFI_EVENT  Event,
  IN VOID       *Context
  )
{
  NET_LIST_FOR_EACH (Entry, &Udp4Service->ChildrenList) {
    //
    // Iterate all the instances belonging to this service context.
    //
    Instance = NET_LIST_USER_STRUCT (Entry, UDP4_INSTANCE_DATA, Link);
    NET_CHECK_SIGNATURE (Instance, UDP4_INSTANCE_DATA_SIGNATURE);

    if (!Instance->Configured || (Instance->ConfigData.ReceiveTimeout == 0)) {
      //
      // Skip this instance if it's not configured or no receive timeout.
      //
      continue;
    }

    NET_LIST_FOR_EACH_SAFE (WrapEntry, NextEntry, &Instance->RcvdDgramQue) {
      //
      // Iterate all the rxdatas belonging to this udp instance.
```

```
      //
      Wrap = NET_LIST_USER_STRUCT (WrapEntry, UDP4_RXDATA_WRAP, Link);

      //
      // TimeoutTick unit is microsecond, MNP_TIMEOUT_CHECK_INTERVAL unit is 100ns.
      //
      if (Wrap->TimeoutTick < (UDP4_TIMEOUT_INTERVAL / 10)) {
        //
        // Remove this RxData if it timeouts.
        //
        Udp4RecycleRxDataWrap (NULL, (VOID *)Wrap);
      } else {
        Wrap->TimeoutTick -= (UDP4_TIMEOUT_INTERVAL / 10);
      }
    }
  }
}
```

Here is `Wrap` the corresponding structure `UDP4_RXDATA_WRAP` :

c AI generated projects 登录复制 run

```
typedef struct _UDP4_RXDATA_WRAP_ {
  LIST_ENTRY              Link;
  NET_BUF                 *Packet;
  UINT32                  TimeoutTick;
  EFI_UDP4_RECEIVE_DATA   RxData;
} UDP4_RXDATA_WRAP;
```

It is `Udp4WrapRxData()` created and then placed in a queue for the UDP driver to process. If it is not processed in time, it will expire. The expiration time is also specified by the member here `TimeoutTick` , which is specified by another value:

c AI generated projects 登录复制 run

```
Wrap->TimeoutTick = Instance->ConfigData.ReceiveTimeout;
```

`Instance` This is `UDP4_INSTANCE_DATA` a configuration parameter that will be introduced later `ConfigData` . `ConfigData.ReceiveTimeout` Its value is -1 when it is created, indicating that it will not expire:

c AI generated projects 登录复制 run

```
  //
  // use the -1 magic number to disable the receiving process of the ip instance.
  //
  Ip4ConfigData->ReceiveTimeout = (UINT32)(-1);
```

However, it can be modified in the UDP4 configuration:

c AI generated projects 登录复制 run

```
EFI_STATUS
EFIAPI
Udp4Configure (
  IN EFI_UDP4_PROTOCOL     *This,
  IN EFI_UDP4_CONFIG_DATA  *UdpConfigData OPTIONAL
  )
{
  if (UdpConfigData != NULL) {
    if (Instance->Configured) {
      //
      // Save the reconfigurable parameters.
      //
      Instance->ConfigData.TransmitTimeout = UdpConfigData->TransmitTimeout;
    }
  }
}
```

UDP4_INSTANCE_DATA

`UDP4_INSTANCE_DATA` Represents a UDP4 sub-item, the others are located in NetworkPkg\Udp4Dxe\Udp4Impl.h:

c AI generated projects 登录复制 run

```
typedef struct _UDP4_INSTANCE_DATA_ {
  UINT32                  Signature;
  LIST_ENTRY              Link;

  UDP4_SERVICE_DATA       *Udp4Service;
  EFI_UDP4_PROTOCOL       Udp4Proto;
  EFI_UDP4_CONFIG_DATA    ConfigData;
  EFI_HANDLE              ChildHandle;
  BOOLEAN                 Configured;
  BOOLEAN                 IsNoMapping;

  NET_MAP                 TxTokens;
  NET_MAP                 RxTokens;

  NET_MAP                 McastIps;

  LIST_ENTRY              RcvdDgramQue;
  LIST_ENTRY              DeliveredDgramQue;

  UINT16                  HeadSum;

  EFI_STATUS              IcmpError;

  IP_IO_IP_INFO           *IpInfo;

  BOOLEAN                 InDestroy;
} UDP4_INSTANCE_DATA;
```

The following are some of the more important members:

- `Udp4Service` : Points to the structure of UDP4 service.
- `Udp4Proto` : Corresponds to EFI_UDP4_PROTOCOL , which will be further introduced later.
- `ConfigData` :UDP configuration data:

c AI generated projects 登录复制 run

```
typedef struct {
  //
  // Receiving Filters
  //
  BOOLEAN             AcceptBroadcast;
```

```
  BOOLEAN             AcceptPromiscuous;
  BOOLEAN             AcceptAnyPort;
  BOOLEAN             AllowDuplicatePort;
  //
  // I/O parameters
  //
  UINT8               TypeOfService;
  UINT8               TimeToLive;
  BOOLEAN             DoNotFragment;
  UINT32              ReceiveTimeout;
  UINT32              TransmitTimeout;
  //
  // Access Point
  //
  BOOLEAN             UseDefaultAddress;
  EFI_IPv4_ADDRESS    StationAddress;
  EFI_IPv4_ADDRESS    SubnetMask;
  UINT16              StationPort;
  EFI_IPv4_ADDRESS    RemoteAddress;
  UINT16              RemotePort;
} EFI_UDP4_CONFIG_DATA;
```

- `TxTokens`, `RxTokens`: Describes the mapping of sending and receiving data:

c AI generated projects 登录复制 run

```c
typedef struct {
  LIST_ENTRY    Used;
  LIST_ENTRY    Recycled;
  UINTN         Count;
} NET_MAP;
```

The real token is `EFI_UDP4_COMPLETION_TOKEN`:

c AI generated projects 登录复制 run

```c
typedef struct {
  EFI_EVENT     Event;
  EFI_STATUS    Status;
  union {
    EFI_UDP4_RECEIVE_DATA     *RxData;
    EFI_UDP4_TRANSMIT_DATA    *TxData;
  } Packet;
} EFI_UDP4_COMPLETION_TOKEN;
```

- `RcvdDgramQue`, `DeliveredDgramQue`: Queues that process sending and receiving data.
- `IpInfo`: The structure required by the underlying IP4 instance:

c AI generated projects 登录复制 run

```c
///
/// The IP_IO_IP_INFO is used in IpIoSend() to override the default IP instance
/// in IP_IO.
///
typedef struct _IP_IO_IP_INFO {
  EFI_IP_ADDRESS               Addr;
  IP_IO_IP_MASK                PreMask;
  LIST_ENTRY                   Entry;
  EFI_HANDLE                   ChildHandle;
  IP_IO_IP_PROTOCOL            Ip;
  IP_IO_IP_COMPLETION_TOKEN    DummyRcvToken;
  INTN                         RefCnt;
  UINT8                        IpVersion;
} IP_IO_IP_INFO;
```

From the comments you can see that it is used when sending data.

EFI_UDP4_PROTOCOL

The interface for UDP4 communication. The structure of the Protocol is as follows:

c AI generated projects 登录复制 run

```c
///
/// The EFI_UDP4_PROTOCOL defines an EFI UDPv4 Protocol session that can be used
/// by any network drivers, applications, or daemons to transmit or receive UDP packets.
/// This protocol instance can either be bound to a specified port as a service or
/// connected to some remote peer as an active client. Each instance has its own settings,
/// such as the routing table and group table, which are independent from each other.
///
struct _EFI_UDP4_PROTOCOL {
  EFI_UDP4_GET_MODE_DATA    GetModeData;
  EFI_UDP4_CONFIGURE        Configure;
  EFI_UDP4_GROUPS           Groups;
  EFI_UDP4_ROUTES           Routes;
  EFI_UDP4_TRANSMIT         Transmit;
  EFI_UDP4_RECEIVE          Receive;
  EFI_UDP4_CANCEL           Cancel;
  EFI_UDP4_POLL             Poll;
};
```

The corresponding implementation:

c AI generated projects 登录复制 run

```c
EFI_UDP4_PROTOCOL  mUdp4Protocol = {
  Udp4GetModeData,
  Udp4Configure,
  Udp4Groups,
  Udp4Routes,
  Udp4Transmit,
  Udp4Receive,
  Udp4Cancel,
  Udp4Poll
};
```

The implementation of these functions will be introduced later.

Udp4.GetModeData

The corresponding implementation is `Udp4GetModeData()`:

```c
EFI_STATUS
EFIAPI
Udp4GetModeData (
  IN  EFI_UDP4_PROTOCOL               *This,
  OUT EFI_UDP4_CONFIG_DATA            *Udp4ConfigData OPTIONAL,
  OUT EFI_IP4_MODE_DATA               *Ip4ModeData    OPTIONAL,
  OUT EFI_MANAGED_NETWORK_CONFIG_DATA  *MnpConfigData  OPTIONAL,
  OUT EFI_SIMPLE_NETWORK_MODE         *SnpModeData    OPTIONAL
  )
{
  if (Udp4ConfigData != NULL) {
    //
    // Set the Udp4ConfigData.
    //
    CopyMem (Udp4ConfigData, &Instance->ConfigData, sizeof (*Udp4ConfigData));
  }

  Ip = Instance->IpInfo->Ip.Ip4;

  //
  // Get the underlying Ip4ModeData, MnpConfigData and SnpModeData.
  //
  Status = Ip->GetModeData (Ip, Ip4ModeData, MnpConfigData, SnpModeData);

}
```

From this we can see that the upper-layer network protocol can obtain all the mode data of the lower layer.

For UDP4, the data is `UDP4_INSTANCE_DATA` in `ConfigData` the members.

Udp4.Configure

The corresponding implementation is `Udp4Configure()`:

```c
EFI_STATUS
EFIAPI
Udp4Configure (
  IN EFI_UDP4_PROTOCOL     *This,
  IN EFI_UDP4_CONFIG_DATA  *UdpConfigData OPTIONAL
  )
{
  // 根据是否有配置存在两种情况，没有数据相当于重置
  if (UdpConfigData != NULL) {
    if (Instance->Configured) {
      //
      // The instance is already configured, try to do the re-configuration.
      //
      if (!Udp4IsReconfigurable (&Instance->ConfigData, UdpConfigData)) {
        //
        // If the new configuration data wants to change some unreconfigurable
        // settings, return EFI_ALREADY_STARTED.
        //
        Status = EFI_ALREADY_STARTED;
        goto ON_EXIT;
      }

      //
      // Save the reconfigurable parameters.
      //
      Instance->ConfigData.TypeOfService   = UdpConfigData->TypeOfService;
      Instance->ConfigData.TimeToLive      = UdpConfigData->TimeToLive;
      Instance->ConfigData.DoNotFragment   = UdpConfigData->DoNotFragment;
      Instance->ConfigData.ReceiveTimeout  = UdpConfigData->ReceiveTimeout;
      Instance->ConfigData.TransmitTimeout = UdpConfigData->TransmitTimeout;
    } else {
      //
      // Construct the Ip configuration data from the UdpConfigData.
      //
      Udp4BuildIp4ConfigData (UdpConfigData, &Ip4ConfigData);

      //
      // Configure the Ip instance wrapped in the IpInfo.
      //
      Status = IpIoConfigIp (Instance->IpInfo, &Ip4ConfigData);
      if (EFI_ERROR (Status)) {
        if (Status == EFI_NO_MAPPING) {
          Instance->IsNoMapping = TRUE;
        }

        goto ON_EXIT;
      }

      Instance->IsNoMapping = FALSE;

      //
      // Save the configuration data.
      //
      CopyMem (&Instance->ConfigData, UdpConfigData, sizeof (Instance->ConfigData));
      IP4_COPY_ADDRESS (&Instance->ConfigData.StationAddress, &Ip4ConfigData.StationAddress);
      IP4_COPY_ADDRESS (&Instance->ConfigData.SubnetMask, &Ip4ConfigData.SubnetMask);

      //
      // Try to allocate the required port resource.
      //
      Status = Udp4Bind (&Udp4Service->ChildrenList, &Instance->ConfigData);
      if (EFI_ERROR (Status)) {
        //
        // Reset the ip instance if bind fails.
        //
        IpIoConfigIp (Instance->IpInfo, NULL);
        goto ON_EXIT;
      }

      //
      // Pre calculate the checksum for the pseudo head, ignore the UDP length first.
      //
      CopyMem (&LocalAddr, &Instance->ConfigData.StationAddress, sizeof (IP4_ADDR));
      CopyMem (&RemoteAddr, &Instance->ConfigData.RemoteAddress, sizeof (IP4_ADDR));
      Instance->HeadSum = NetPseudoHeadChecksum (
                            LocalAddr,
                            RemoteAddr,
                            EFI_IP_PROTO_UDP,

```

```
                          0
                          );

      Instance->Configured = TRUE;
    }
  } else {
    //
    // UdpConfigData is NULL, reset the instance.
    //
    Instance->Configured  = FALSE;
    Instance->IsNoMapping = FALSE;

    //
    // Reset the Ip instance wrapped in the IpInfo.
    //
    IpIoConfigIp (Instance->IpInfo, NULL);

    //
    // Cancel all the user tokens.
    //
    Instance->Udp4Proto.Cancel (&Instance->Udp4Proto, NULL);

    //
    // Remove the buffered RxData for this instance.
    //
    Udp4FlushRcvdDgram (Instance);
  }
}
```

Depending on the input parameters and whether it has been configured, different processes will be followed. In addition, UDP4 will also configure the interface that further calls IP4 for configuration.

Udp4.Transmit

The corresponding implementation is `Udp4Transmit()`:

c — AI generated projects — 登录复制 — run

```c
EFI_STATUS
EFIAPI
Udp4Transmit (
  IN EFI_UDP4_PROTOCOL          *This,
  IN EFI_UDP4_COMPLETION_TOKEN  *Token
  )
{
  //
  // Validate the Token, if the token is invalid return the error code.
  //
  Status = Udp4ValidateTxToken (Instance, Token);
  if (EFI_ERROR (Status)) {
    goto ON_EXIT;
  }

  if (EFI_ERROR (NetMapIterate (&Instance->TxTokens, Udp4TokenExist, Token)) ||
      EFI_ERROR (NetMapIterate (&Instance->RxTokens, Udp4TokenExist, Token)))
  {
    //
    // Try to find a duplicate token in the two token maps, if found, return
    // EFI_ACCESS_DENIED.
    //
    Status = EFI_ACCESS_DENIED;
    goto ON_EXIT;
  }

  TxData = Token->Packet.TxData;

  //
  // Create a net buffer to hold the user buffer and the udp header.
  //
  Packet = NetbufFromExt (
             (NET_FRAGMENT *)TxData->FragmentTable,
             TxData->FragmentCount,
             UDP4_HEADER_SIZE,
             0,
             Udp4NetVectorExtFree,
             NULL
             );
  if (Packet == NULL) {
    Status = EFI_OUT_OF_RESOURCES;
    goto ON_EXIT;
  }

  //
  // Store the IpIo in ProtoData.
  //
  Udp4Service                       = Instance->Udp4Service;
  *((UINTN *)&Packet->ProtoData[0]) = (UINTN)(Udp4Service->IpIo);

  Udp4Header = (EFI_UDP_HEADER *)NetbufAllocSpace (Packet, UDP4_HEADER_SIZE, TRUE);
  ASSERT (Udp4Header != NULL);

  ConfigData = &Instance->ConfigData;

  //
  // Fill the udp header.
  //
  Udp4Header->SrcPort  = HTONS (ConfigData->StationPort);
  Udp4Header->DstPort  = HTONS (ConfigData->RemotePort);
  Udp4Header->Length   = HTONS ((UINT16)Packet->TotalSize);
  Udp4Header->Checksum = 0;

  UdpSessionData = TxData->UdpSessionData;
  IP4_COPY_ADDRESS (&Override.Ip4OverrideData.SourceAddress, &ConfigData->StationAddress);

  if (UdpSessionData != NULL) {
    //
    // Set the SourceAddress, SrcPort and Destination according to the specified
    // UdpSessionData.
    //
    if (!EFI_IP4_EQUAL (&UdpSessionData->SourceAddress, &mZeroIp4Addr)) {
      IP4_COPY_ADDRESS (&Override.Ip4OverrideData.SourceAddress, &UdpSessionData->SourceAddress);
    }

    if (UdpSessionData->SourcePort != 0) {
      Udp4Header->SrcPort = HTONS (UdpSessionData->SourcePort);
    }

```

```
    if (UdpSessionData->DestinationPort != 0) {
      Udp4Header->DstPort = HTONS (UdpSessionData->DestinationPort);
    }

    CopyMem (&Source, &Override.Ip4OverrideData.SourceAddress, sizeof (IP4_ADDR));
    CopyMem (&Destination, &UdpSessionData->DestinationAddress, sizeof (IP4_ADDR));

    //
    // calculate the pseudo head checksum using the overridden parameters.
    //
    HeadSum = NetPseudoHeadChecksum (
                Source,
                Destination,
                EFI_IP_PROTO_UDP,
                0
                );
  } else {
    //
    // UdpSessionData is NULL, use the address and port information previously configured.
    //
    CopyMem (&Destination, &ConfigData->RemoteAddress, sizeof (IP4_ADDR));

    HeadSum = Instance->HeadSum;
  }

  //
  // calculate the checksum.
  //
  Udp4Header->Checksum = Udp4Checksum (Packet, HeadSum);
  if (Udp4Header->Checksum == 0) {
    //
    // If the calculated checksum is 0, fill the Checksum field with all ones.
    //
    Udp4Header->Checksum = 0xffff;
  }

  //
  // Fill the IpIo Override data.
  //
  if (TxData->GatewayAddress != NULL) {
    IP4_COPY_ADDRESS (&Override.Ip4OverrideData.GatewayAddress, TxData->GatewayAddress);
  } else {
    ZeroMem (&Override.Ip4OverrideData.GatewayAddress, sizeof (EFI_IPv4_ADDRESS));
  }

  Override.Ip4OverrideData.Protocol      = EFI_IP_PROTO_UDP;
  Override.Ip4OverrideData.TypeOfService = ConfigData->TypeOfService;
  Override.Ip4OverrideData.TimeToLive    = ConfigData->TimeToLive;
  Override.Ip4OverrideData.DoNotFragment = ConfigData->DoNotFragment;

  //
  // Save the token into the TxToken map.
  //
  Status = NetMapInsertTail (&Instance->TxTokens, Token, Packet);

  //
  // Send out this datagram through IpIo.
  //
  IpDestAddr.Addr[0] = Destination;
  Status             = IpIoSend (
                         Udp4Service->IpIo,
                         Packet,
                         Instance->IpInfo,
                         Instance,
                         Token,
                         &IpDestAddr,
                         &Override
                         );
}
```

Udp4.Receive

The corresponding implementation is `Udp4Receive()`:

c — AI generated projects — 登录复制 — run

```
EFI_STATUS
EFIAPI
Udp4Receive (
  IN EFI_UDP4_PROTOCOL          *This,
  IN EFI_UDP4_COMPLETION_TOKEN  *Token
  )
{
  if (EFI_ERROR (NetMapIterate (&Instance->RxTokens, Udp4TokenExist, Token)) ||
      EFI_ERROR (NetMapIterate (&Instance->TxTokens, Udp4TokenExist, Token)))
  {
    //
    // Return EFI_ACCESS_DENIED if the specified token is already in the TxTokens or
    // RxTokens map.
    //
    Status = EFI_ACCESS_DENIED;
    goto ON_EXIT;
  }

  Token->Packet.RxData = NULL;

  //
  // Save the token into the RxTokens map.
  //
  Status = NetMapInsertTail (&Instance->RxTokens, Token, NULL);
  if (EFI_ERROR (Status)) {
    Status = EFI_NOT_READY;
    goto ON_EXIT;
  }

  //
  // If there is an icmp error, report it.
  //
  Udp4ReportIcmpError (Instance);

  //
  // Try to deliver the received datagrams.
  //
  Udp4InstanceDeliverDgram (Instance);

  //
```

```
  // Dispatch the DPC queued by the NotifyFunction of Token->Event.
  //
  DispatchDpc ();
}
```

Just like Receive in other network protocols, the focus is on registering the Token.

Udp4.Poll

The corresponding implementation is that `Udp4Poll()` its code implementation is to call the next layer of Poll:

c AI generated projects 登录复制 run

```c
EFI_STATUS
EFIAPI
Udp4Poll (
  IN EFI_UDP4_PROTOCOL  *This
  )
{
  Ip       = Instance->IpInfo->Ip.Ip4;
  //
  // Invode the Ip instance consumed by the udp instance to do the poll operation.
  //
  return Ip->Poll (Ip);
}
```

Code Sample

DHCP and DNS all use UDP, which will be explained further later.