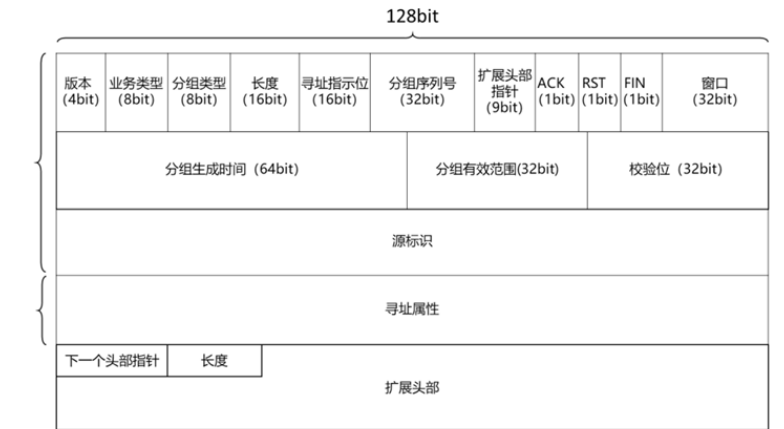


# NDN-DPDK 项目解析

## 构造网络中传输的包

下面教程是讲解如何在原有 NDN-DKDK 项目中的兴趣包结构的基础上构造出如下图所示的包：



## 一、添加编码类型

包在网络中传输之前要进行 TLV 编码，编码的最小字段长度为 1 个字节，也就是 8bit，所以在后面编码时对上面的包结构进行了小幅度修改：

- 版本改为 8bit 长度
- 扩展头部指针改为 8bit 长度
- 删除 ACK、RST、FIN 这三个字段

下面是各个字段对应的类型 (Type) 的编号(写到 `ndn/an/tlv-type.go` 文件中)

1、版本	TtVersion	= 0x0800
2、业务类型	TtServiceType	= 0x0804
3、分组类型	TtPacketType	= 0x0808
4、长度	TtLength	= 0x080c
5、寻址指示位	TtAddressingIndicator	= 0x0810
6、分组序列号	TtPacketSequenceNumber	= 0x0814
7、扩展头部指针	TtExtensionHeaderPointer	= 0x0818
8、窗口	TtWindow	= 0x081c
9、分组生成时间	TtPacketCreationTime	= 0x0820
10、分组有效范围	TtPacketValidityRange	= 0x0824
11、校验位	TtChecksum	= 0x0828
12、源标识	TtSourceIdentifier	= 0x082c
13、寻址属性	TtAddressingAttributes	= 0x0830
14、扩展头部	TtExtensionHeader	= 0x0834

编号的规律是编号的二进制后两位为 0

## 二、修改兴趣包

打开 `ndn-dpdk/ndn/interest.go` 文件

修改 **Interest** 结构体

添加上述 14 个字段

18	// Interest represents an Interest packet.	Version	[]byte
19	type Interest struct {	ServiceType	[]byte
20	packet       *Packet	PacketType	[]byte
21	Name         Name	Length	[]byte
22	CanBePrefix bool	AddressingIndicator	[]byte
23	MustBeFresh bool	PacketSequenceNumber	[]byte
24	ForwardingHint ForwardingHint	ExtensionHeaderPointer	[]byte
25	Nonce        Nonce	Window	[]byte
26	Lifetime     time.Duration	PacketCreationTime	[]byte
27	HopLimit     HopLimit	PacketValidityRange	[]byte
28	AppParameters []byte	Checksum	[]byte
29	SigInfo       *SigInfo	SourceIdentifier	[]byte
30	SigValue      []byte	AddressingAttributes	[]byte
31	Version       []byte	ExtensionHeader	[]byte
32	ServiceType   []byte		
33	PacketType    []byte		
34	Length        []byte		
35	AddressingIndicator []byte		
36	PacketSequenceNumber []byte		
37	ExtensionHeaderPointer []byte		
38	Window        []byte		
39	PacketCreationTime []byte		
40	PacketValidityRange []byte		
41	Checksum       []byte		
42	SourceIdentifier []byte		
43	AddressingAttributes []byte		
44	ExtensionHeader []byte		
45	}		

修改 Field()方法，将添加的 14 个字段编码为 TLV 格式，并将其添加到字段列表中。

```

if len(interest.Version) > 0 {
    fields = append(fields, tlv.TLVBytes(an.TtVersion, interest.Version))
}

if len(interest.ServiceType) > 0 {
    fields = append(fields, tlv.TLVBytes(an.TtServiceType, interest.ServiceType))
}

func (interest Interest) Field() tlv.Field {
    if len(interest.Version) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtVersion, interest.Version))
    }
    if len(interest.ServiceType) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtServiceType, interest.ServiceType))
    }
    if len(interest.PacketType) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtPacketType, interest.PacketType))
    }
    if len(interest.Length) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtLength, interest.Length))
    }
    if len(interest.AddressingIndicator) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtAddressingIndicator, interest.AddressingIndicator))
    }
    if len(interest.PacketSequenceNumber) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtPacketSequenceNumber, interest.PacketSequenceNumber))
    }
    if len(interest.ExtensionHeaderPointer) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtExtensionHeaderPointer, interest.ExtensionHeaderPointer))
    }
    if len(interest.Window) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtWindow, interest.Window))
    }
    if len(interest.PacketCreationTime) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtPacketCreationTime, interest.PacketCreationTime))
    }
    if len(interest.PacketValidityRange) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtPacketValidityRange, interest.PacketValidityRange))
    }
    if len(interest.Checksum) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtChecksum, interest.Checksum))
    }
    if len(interest.SourceIdentifier) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtSourceIdentifier, interest.SourceIdentifier))
    }
    if len(interest.AddressingAttributes) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtAddressingAttributes, interest.AddressingAttributes))
    }
    if len(interest.ExtensionHeader) > 0 {
        fields = append(fields, tlv.TLVBytes(an.TtExtensionHeader, interest.ExtensionHeader))
    }
}

```

更新 UnmarshalBinary() 方法：在解码函数中，添加对新字段的 TLV 解码处理。

```

273 func (interest *Interest) UnmarshalBinary(wire []byte) (e error) {
304     // 添加对自定义字段的解析
305     case an.TtVersion:
306         interest.Version = de.Value
307     case an.TtServiceType:
308         interest.ServiceType = de.Value
309     case an.TtPacketType:
310         interest.PacketType = de.Value
311     case an.TtLength:
312         interest.Length = de.Value
313     case an.TtAddressingIndicator:
314         interest.AddressingIndicator = de.Value
315     case an.TtPacketSequenceNumber:
316         interest.PacketSequenceNumber = de.Value
317     case an.TtExtensionHeaderPointer:
318         interest.ExtensionHeaderPointer = de.Value
319     case an.TtWindow:
320         interest.Window = de.Value
321     case an.TtPacketCreationTime:
322         interest.PacketCreationTime = de.Value
323     case an.TtPacketValidityRange:
324         interest.PacketValidityRange = de.Value
325     case an.TtChecksum:
326         interest.Checksum = de.Value
327     case an.TtSourceIdentifier:
328         interest.SourceIdentifier = de.Value
329     case an.TtAddressingAttributes:
330         interest.AddressingAttributes = de.Value
331     case an.TtExtensionHeader:
332         interest.ExtensionHeader = de.Value
333     case an.TtAppParameters:
334         interest.AppParameters = de.Value
335         paramsPortion = de.WireAfter()
336     case an.TtSigInfo:
337         var si SigInfo
338         if e := de.UnmarshalValue(&si); e != nil {
339             return e

```

case an.TtVersion:

interest.Version = de.Value

case an.TtServiceType:

interest.ServiceType = de.Value

case an.TtPacketType:

interest.PacketType = de.Value

case an.TtLength:

interest.Length = de.Value

case an.TtAddressingIndicator:

interest.AddressingIndicator = de.Value

case an.TtPacketSequenceNumber:

interest.PacketSequenceNumber = de.Value

case an.TtExtensionHeaderPointer:

interest.ExtensionHeaderPointer = de.Value

case an.TtWindow:

interest.Window = de.Value

case an.TtPacketCreationTime:

interest.PacketCreationTime = de.Value

case an.TtPacketValidityRange:

interest.PacketValidityRange = de.Value

case an.TtChecksum:

interest.Checksum = de.Value

case an.TtSourceIdentifier:

interest.SourceIdentifier = de.Value

case an.TtAddressingAttributes:

```
interest.AddressingAttributes = de.Value
case an.TtExtensionHeader:
interest.ExtensionHeader = de.Value
```

### 三、填充包内容

打开 cmd/ndndpdk-godemo/ping.go 修改里面的代码

对新添的 14 个字段进行填充, 根据每个字段的长度, 每 8bit 填充一个数字。

例如:

Version 长度为 8bit 填充一个数字“1”

ServiceType 长度为 8bit 填充一个数字“2”

AddressingIndicator 长度为 16bit 填充两个数字“44”

PacketCreationTime 长度为 64bit 填充 8 个数字“99999999”

```

    oooooo
75 func init() {
121     for {
122         select {
123         case <-c.Context.Done():
124             return nil
125         case timestamp := <-ticker.C:
126             go func(t0 time.Time, s uint64) {
127
128                 interest := ndn.MakeInterest(fmt.Sprintf("%s/%016X", name, s), ndn.MustBeFreshFlag, lifetime)
129                 interest.Version = []byte("1")
130                 interest.ServiceType = []byte("2")
131                 interest.PacketType = []byte("3")
132                 interest.Length = []byte("44")
133                 interest.AddressingIndicator = []byte("55")
134                 interest.PacketSequenceNumber = []byte("6666")
135                 interest.ExtensionHeaderPointer = []byte("7")
136                 interest.Window = []byte("8888")
137                 interest.PacketCreationTime = []byte("99999999")
138                 interest.PacketValidityRange = []byte("1010")
139                 interest.Checksum = []byte("1111")
140                 interest.SourceIdentifier = []byte("2222222222222222")
141                 interest.AddressingAttributes = []byte("3333333333333333")
142                 interest.ExtensionHeader = []byte("4444444444444444")
143                 e := endpoint.Consume(ctx, interest, endpoint.ConsumerOptions{
144                     Verifier: verifier,
145                 })
146                 rtt := time.Since(t0)
147                 if e == nil {
148                     nDataL, nErrorsL := nData.Add(1), nErrors.Load()
149                     log.Printf("%6.2f%% D %016X %6dus", 100*float64(nDataL)/float64(nDataL+nErrorsL), s, rtt.M)
150                 } else {
151                     nDataL, nErrorsL := nData.Load(), nErrors.Add(1)
152                     log.Printf("%6.2f%% E %016X %v", 100*float64(nDataL)/float64(nDataL+nErrorsL), s, e)
153                 }
154             }(timestamp, seqNum)
155             seqNum++
156         }
157     }
158 }
159 })
160 }
```

### 四、添加测试代码

#### 1、查看 TLV 编码后的包内容

打开 ndn/l3/face.go

在 face 发送包之前添加打印语句, 显示包的编码内容

```
fmt.Println("Wire: ", wire)
```

```

151 func (f *face) txLoop() {
152     for l3packet := range f.tx {
153         pkt := l3packet.ToPacket()
154         frames, e := f.fragmenter.Fragment(pkt)
155         if e != nil {
156             continue
157         }
158
159         for _, frame := range frames {
160             if wire, e := tlv.EncodeFrom(frame); e == nil {
161                 fmt.Println("Wire: ", wire)
162                 fmt.Println("[face.txLoop()]send pkt")
163                 f.faceTr.Write(wire)
164             }
165         }
166     }
167     f.faceTr.Close()
168 }

```

## 2、查看接收端解析的内容

打开 ndn/endpoint/producer.go

修改 handleInteres 方法，打印输出相关字段的内容

```

fmt.Println("Interest.Version: ", interest.Version)
fmt.Println("Interest.ServiceType: ", interest.ServiceType)
fmt.Println("Interest.PacketType: ", interest.PacketType)
...

```

```

136 func (p *producer) handleInterest(ctx context.Context, wg *sync.WaitGroup, pkt *ndn.Packet) {
137     defer wg.Done()
138     interest := pkt.Interest
139     fmt.Println("Interest.Version: ", interest.Version)
140     fmt.Println("Interest.ServiceType: ", interest.ServiceType)
141     fmt.Println("Interest.PacketType: ", interest.PacketType)
142     fmt.Println("Interest.Length: ", interest.Length)
143     fmt.Println("Interest.AddressingIndicator: ", interest.AddressingIndicator)
144     fmt.Println("Interest.PacketSequenceNumber: ", interest.PacketSequenceNumber)
145     fmt.Println("Interest.ExtensionHeader: ", interest.ExtensionHeader)
146     if !p.Prefix.IsPrefixOf(interest.Name) {
147         return
148     }
149
150     ctx1, cancel1 := context.WithTimeout(ctx, interest.ApplyDefaultLifetime())
151     defer cancel1()
152     data, e := p.Handler(ctx1, *interest)
153
154     var reply *ndn.Packet
155     if e != nil {
156         if nackError, ok := e.(producerNackError); ok {
157             nack := ndn.MakeNack(interest, uint8(nackError))
158             reply = nack.ToPacket()
159         }
160     } else if data.CanSatisfy(*interest) {
161         if (data.SigInfo == nil || data.SigInfo.Type == an.SigNull) && p.DataSigner != nil {
162             if e := p.DataSigner.Sign(&data); e != nil {

```

## 五、编译运行

创建两个虚拟机：虚拟机 A、虚拟机 B

虚拟机 A 作为 client 发送兴趣包

虚拟机 B 作为 server 接收兴趣包并解析

按前面的步骤修改完代码后开始编译：

```
cd /home/lwj/Desktop/ndn-dpdk
```

```
NDNDPDK_MK_RELEASE=1 make
```

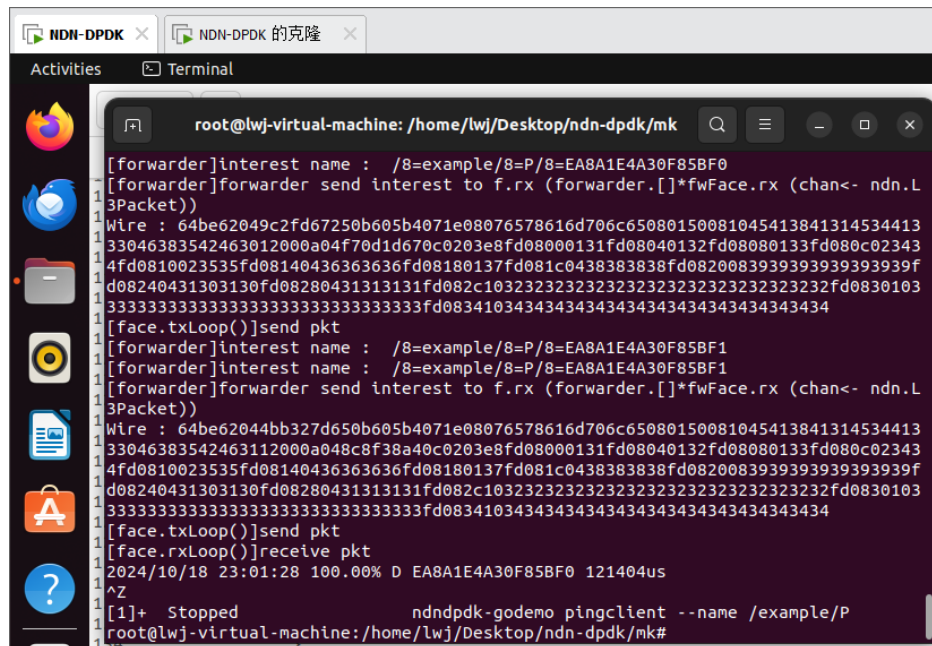
```
cd mk/
```

```
sudo ./install.sh
```

编译完成后的运行步骤可以参考 NDN-DPDK 项目解析三中的转发器实验

下面是运行结果：

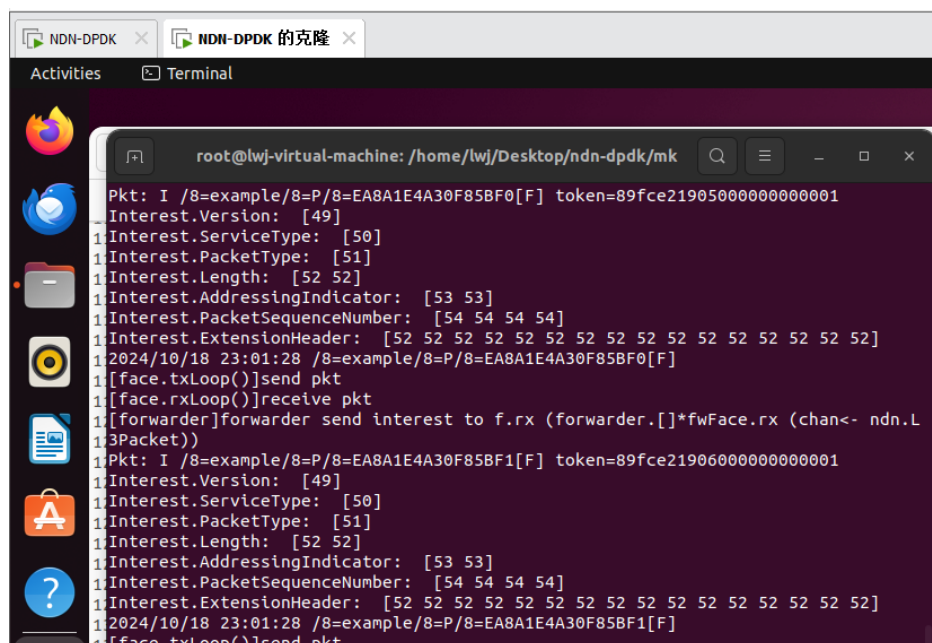
主机 A client 发送兴趣包



```
root@lwj-virtual-machine: /home/lwj/Desktop/ndn-dpdk/mk

[forwarder]interest name : /8=example/8=P/8=EA8A1E4A30F85BF0
[forwarder]forwarder send interest to f.rx (forwarder.[]*fwFace.rx (chan<- ndn.L
3Packet))
1 Wire : 64be62049c2fd67250b605b4071e08076578616d706c65080150081045413841314534413
33046383542463012000a04f70d1d670c0203e8fd08000131fd08040132fd08080133fd080c02343
4fd0810023535fd081404363636fd08180137fd081c0438383838fd0820083939393939393939f
d08240431303130fd08280431313131fd082c10323232323232323232323232323232fd0830103
33333333333333333333333333333333fd08341034343434343434343434343434343434343434
1 [face.txLoop()]send pkt
1 [forwarder]interest name : /8=example/8=P/8=EA8A1E4A30F85BF1
1 [forwarder]interest name : /8=example/8=P/8=EA8A1E4A30F85BF1
1 [forwarder]forwarder send interest to f.rx (forwarder.[]*fwFace.rx (chan<- ndn.L
3Packet))
1 Wire : 64be62044bb327d650b605b4071e08076578616d706c65080150081045413841314534413
33046383542463112000a048c8f38a40c0203e8fd08000131fd08040132fd08080133fd080c02343
4fd0810023535fd08140436363636fd08180137fd081c0438383838fd0820083939393939393939f
d08240431303130fd08280431313131fd082c10323232323232323232323232323232fd0830103
33333333333333333333333333333333fd08341034343434343434343434343434343434343434
1 [face.txLoop()]send pkt
1 [face.rxLoop()]receive pkt
1 2024/10/18 23:01:28 100.00% D EA8A1E4A30F85BF0 121404us
1 ^Z
1 [1]+ Stopped ndndpdk-godemo pingclient --name /example/P
root@lwj-virtual-machine: /home/lwj/Desktop/ndn-dpdk/mk#
```

主机 B 接收解析兴趣包，打印输出部分相关字段



```
root@lwj-virtual-machine: /home/lwj/Desktop/ndn-dpdk/mk

Pkt: I /8=example/8=P/8=EA8A1E4A30F85BF0[F] token=89fce219050000000000001
Interest.Version: [49]
1 Interest.ServiceType: [50]
1 Interest.PacketType: [51]
1 Interest.Length: [52 52]
1 Interest.AddressingIndicator: [53 53]
1 Interest.PacketSequenceNumber: [54 54 54 54]
1 Interest.ExtensionHeader: [52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52]
1 2024/10/18 23:01:28 /8=example/8=P/8=EA8A1E4A30F85BF0[F]
1 [face.txLoop()]send pkt
1 [face.rxLoop()]receive pkt
1 [forwarder]forwarder send interest to f.rx (forwarder.[]*fwFace.rx (chan<- ndn.L
3Packet))
1 Pkt: I /8=example/8=P/8=EA8A1E4A30F85BF1[F] token=89fce219060000000000001
1 Interest.Version: [49]
1 Interest.ServiceType: [50]
1 Interest.PacketType: [51]
1 Interest.Length: [52 52]
1 Interest.AddressingIndicator: [53 53]
1 Interest.PacketSequenceNumber: [54 54 54 54]
1 Interest.ExtensionHeader: [52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52]
1 2024/10/18 23:01:28 /8=example/8=P/8=EA8A1E4A30F85BF1[F]
1 [face.txLoop()]send pkt
```

从主机 A 的输出中我们可以看到兴趣包被编码成为下面的字段：

64be62044bb327d650b605b4071e08076578616d706c6508015008104541384131453441333046383542

