目的

处理因arp cache刷新导致的UDP点对点发送出现的丢包情况

设计思路

单纯的滑动窗口 + 重传机制无法高时效性的应对此情况，因为发现之前丢失报文时，可能已经过去很久，时效性不够，如丢失1、2报文，那么等收到3报文时，才会发现之前的报文丢失；

所以更改为在新发送报文前，连续发送10个udp（具体多少个udp，可能需要根据服务器对应丢包数量调节）报文，对端只要收到其中一个即回复“OK”（无视后续的udp报文，设置标记位，等收满10个报文后，重置标记位），本机端标记为可以发送，则允许正式发送；

此发送报文采用单独的fd，单独的端口，此端口收发共用；

考虑到每次发送前均发送10个无关udp报文，较为影响业务报文的发送效率，所以可设计为定时任务，经过一段时间，如5min（因为Linux下默认的arp cache重置时间为10min），主动触发发送，收到回复后，标记为可以发送；或者记录上次收到回复时间戳，等下次udp函数调用时，检查经过的时间，若超过要求时长，则触发无关udp发送

> 比较：
>
> 1)定时任务：似乎与发送频率关联不大，但是如果在netifi中实现，似乎较为复杂
>
> 2)发送前检查时间：若业务udp发送频率较低，可能每次发送前都需要发送10个无关udp；若发送频率较高，则较为高效，且实现较为简单

问题：发送业务报文侧如何感知此标记，全局变量？两者应该都在主线程中，因为netifi中除了接收由threadframe在另一线程处理，其余发送均在主线程

因为是udp点对点发送，那么当对应有多台目标节点，应该为每个目标节点都实施上述行为，故参照已有的点对点UDP map `map<WORD, string*> m_mapDevCodeIp;` 记录为 `map[udp_fd, bool]` / `map[ip, bool]` 其中key表示对端ip（因为此类型报文通过统一的port，所以无需关心），value代表于此对端ip之间的业务udp是否允许直接发送

涉及报文结构 //待敲定

发送报文结构：

```
typedef  struct
{
  //目标主机ip
  //源ip
  int sec;//发出时的时间戳
  int usec;
  unsigned char count; // 标记第几个udp报文，最多10个
}CheckPacketHead;
```

回复报文结构：

```
typedef  struct
{
  int sec;//发出时的时间戳
  int usec;
}CheckReplyPacketHead;
```

计划新增函数或变量：

1）发送报文、接收报文

2）发送方：发送前的处理函数

首先检查上次收到回复报文的时间点与当前时间点的差，是否超出限定时间，如果是，则开始循环发送10个udp报文，每个报文内部count递增，Sleep等待回复（至少10个RTT？会不会太久）；检查对应map中此对端节点间是否允许发送业务udp报文，若仍不允许发送，说明两端连接有问题（因为发送10个udp都没有收到），直接报错返回

3）接收方：接收后的回复函数

收到第一个报文后，即回复（直接SendTo）；~~修改标记位为[已回复]，~~对之后收到的报文，仍然回复~~，等收到count为10的报文时，重置标记位~~

> 因为接收方的回复消息，同样有可能因arp问题丢包！！所以因对每个收到的udp均回复

4）发送方：收到接收方的回复报文后的处理函数

因为接收的处理线程与主线程相互独立，修改对应map（？？此map设计为static？）；收到第一个回复即修改

5）点对点udp主机间是否允许直接发送的map -- static？？

~~6）接收方标记是否收到udp的标记（即对上述的10个udp只回复一次）~~

7）发送方上次收到回复报文的时间点

8）单独的udp fd

其他

需要向threadframe中注册此udp fd的接收处理函数