**修改**

**Bundle.h**

加了一堆bundle的关于地图，area,neighbor的变量

**Bundle.cc**

init()

在Bundle.h中增加的变量，都在init()中初始化

**二进制文件->二进制流->BundlePayLoad**

fstream fin;

fin.open(historyAreaFilePath.c\_str(),ios::in|ios::binary);

fin.seekg(0, ios::end);

int len = fin.tellg();

fin.seekg(0, ios::beg);

u\_char \*resource = new u\_char[len];

fin.read((char \*)resource, len);

fin.close();

fstream fon;

fon.open("./logDocuments/test",ios::out|ios::binary|ios::trunc);

fon.write((char \*)resource,len);

fon.close();\*/

**邻居发现**

两种邻居发现，一种是DTN2本身的基于IP的发现，实现在/servlib/discovery/IPDiscovery。在run()函数中不断地发送hello包，并接收返回的hello包并处理

一种是基于AODV，目前没实现

**关于AODV**

利用ip发现邻居是发现物理上的一跳的邻居，adov是发现物理上的多跳的邻居。无论用的是ip，还是AODV，都使用的是模拟环境，模拟环境是为了判断两个邻居是否能连通的上。是否基于模拟环境或真实环境是有AODV决定的，不是由模拟环境决定的。

aodv将tcp协议栈中的数据包钩出来，并且将符合它的要求的才会传到上层，不符合要求的不会传到上层，dtn则收不到。对于dtn而言，AODV是透明的。

**如何编译AODV**

见文档 编译依赖ubuntu的iaodv

**邻居表的维护过程**

当发现一个邻居后，先处理handle\_link\_created事件

，再处理handle\_contact\_up事件。（这些事件都在GeoHistoryRouter中实现了重载）

在handle\_contact\_up事件中，先查看AreaManager中存储所有邻居的neighbourlist中是否有该neighbour，如果没有，则生成一个neighbour（生成时，将该neighbour的频率向量加入到FrequencyManager中，用来衰减）并加入到neighbourlist。无论有没有该neighbour，都将neighbour的所有频率向量的该时段加一。最后，将该neighbour的频率向量加入到TimeManager中，用来维护当前邻居的频率向量的加一操作。

**交换移动表**

接下来，执行发送给邻居移动表的功能。

还是在handle\_contact\_up中，打开移动表historyarea，检查是否存在，如果存在，则生成一个事件，由GeoHistoryRouter处理。GeoHistoryRouter接受到该事件后，执行handle\_sendBundle()。handle\_sendBundle()再执行sendMessage()。sendMessage()是修改applib/APIServer.cc中的handle\_send()，具体的作用是生成一个bundle，最终生成BundleReceivedEvent事件，GeohistoryRouter重载handle\_bundle\_received事件，接受BundleReceivedEvent事件。最后交给GeohistoryRouter重载的route\_bundle函数，route\_bundle函数又交给route\_neighbourArea\_bundle函数，该函数找到移动表目的地匹配的特定link，发给该link。

**处理接受到的邻居移动表**

重载handle\_bundle\_received()

如果接受到的bundle类型是NEI\_AREA\_BUNDLE且目的地址是本结点，则确定是邻居发送给本结点的移动表。取得该bundle中的payload文件，交给该邻居的nei->getNeighbourArea()->Payload\_update()处理。

Payload\_update()先将payload转化为流，再将这个流保存为文件(./logDocuments/geoHistory\_dtn/neighbour\_area/邻居ip)

再将该文件读取成多个Area，保存在NeighbourArea的成员areaMap中。

做完以上处理后，其实也陆续走了route\_bundle, route\_neighbourArea\_bundle函数，但是并没有什么处理，可以忽视