EXata 扩展（八）：网络层

目标：了解网络层原理，练习添加单播路由协议、组播路由协议、队列模型和调度器模型。

参考：Programmer Guide，4.4 Network Layer

1. 网络层概述

EXata 的网络层包括大量的协议和模型，大致分为以下几类：

- 网络协议
- 路由协议
  - 单播（Unicast）
  - 多播（Multicast）
- 队列模型
- 调度器模型（算法）

1.1 网络协议

网络协议负责数据的转发。EXata 5.1 支持的网络协议有：

- Fixed Comms：Fixed Communications Model: 实现最小的丢包率，可选的固定延迟模型。
- ICMP
- ICMPv6
- IGMP：Internet Group Management Protocol
- Dual IP
- IPv4
- IPv6
- Mobile IPv4
- **NDP：Neighbor Discovery Protocol**

1.2 路由协议

路由协议的主要职能：

- 路由算法确定一个最优路径，并保存到路由器的路由表中。
- 路由器间通过交换消息，来维护路由表。

总的来说，路由协议分为单播路由协议和组播路由协议，针对有线网络（wired networks）、无线网络（wireless networks）或者 混合网络（mixed networks），路由协议也有不同。

1.2.1 单播路由（unicast routing）

注意：有些路由协议实现在应用层，比如Bellman-Ford、OLSR等。

单播路由即只有一个输出接口。针对下层网络的不同，往往可以有不同的选择：

- **Wireless Ad Hoc Networks**
  - 由于拓扑变化频繁，尽量采用On-demand或 reactive 类型的路由协议， 比如 AODV、DSR、LAR1等。
  - 有些主动式（proactive）路由协议也适用 ad hoc 网络，比如 **OLSR，它在应用层实现**。
- **Wired Networks**
  - 有线网络拓扑相对稳定
- **Mixed Networks**
  - AODV、DYMO、OSPFv2、OSPFv3可以跨越混合网络。

EXata5.1 支持的网络层Unicast 路由协议如下表：

Unicast Routing Protocol	Description	Type	Model library
AODV	Ad-hocOn-demand Distance Vector	On-demand	wireless
DSR	Dynamic Source Routing （DSR)	On-demand	wireless
DYMO	Dynamic MANET On-demand	On-demand	wireless
FSRL	Landmark Ad-hocRouting （LANMAR) 在本地采用 Fisheye	Proactive	wireless
IARP	IntrA-zone Routing Protocol ZRP 的分支	Proactive	wireless
IERP	Inter-zone Routing Protocol ZRP 的分支	On-demand	wireless
LAR1	Location-Aided Routing 利用位置信息辅助路由	On-demand	wireless
OSPFv2			
OSPFv3			
STAR	Source Tree Adaptive Routing	Proactive	Wireless
ZRP	Zone Routing Protocol	Hybrid	Wireless

1.2.2 Multicast Routing

- **Wireless Networks**
  - 推荐采用 **ODMRP**，可以减小信道开销，提升可扩展性。
- **Wired Networks**
  - 有线网络的组播路由协议通常构建一个基于源的组播传递树（source-based multicast delivery tree）。

EXata 5.1 目前支持的组播路由协议如下表：

Multicast Routing Protocol	Description	Type	Model Library
DVMRP	Distance Vector Multiple Routing Protocol，为有线网络设计，基于树的组播方案	proactive	Multimedia and Enterprise
MOSPF	Multicast Open Shortest First，OSPF的扩展版本	Proactive	Multimedia and Enterprise
ODMRP	On-Demand Multicast Routing Protocol，基于网状无线ad hoc 的路由协议，适用单个子网	On-demand	wireless
PIM	Protocol Independent Multicast	Proactive	Multimedia and Enterprise

1.2.3 Queues

队列模型在Developer 库中实现。EXata 5.1 支持的队列模型有：

- **FIFIO**：First In First Out
- **RED**：Random Early Drop
- **RIO**：RED with In/Out Bit
- **WRED**：Weighted RED

1.2.4 Schedulers

不同的调度算法决定包或队列处理的顺序。EXata 5.1 支持的调度器有以下：除了DIFFSERV在Multimedia and Enterprise 库实现，其他都在 Developer 库实现。

- CBQ：Class-based Queuing Algorithm，队列分为不同的等级（class），分配不同的带宽，根据可用带宽进行调度。
- DIFFSERV：Differentiated Services （DiffServ）QoS protocol。

- WEIGHTED-FAIR
- WEIGHTED-ROUND-ROBIN

1.3 网络层的常用数据结构

a. **NetworkRoutingProtocolType**：枚举所有网络层协议和所有的路由协议，定义在 network.h. 网络协议采用 NETWORK_PROTOCOL_XXXX 命名，路由协议采用 ROUTING_PROTOCOL_XXX 来命名。

```
// /**
// ENUM        :: NetworkRoutingProtocolType
// DESCRIPTION :: Enlisted different network/routing protocol
// **/
enum NetworkRoutingProtocolType
{
    NETWORK_PROTOCOL_IP = 0,
    NETWORK_PROTOCOL_IPV6,
    NETWORK_PROTOCOL_MOBILE_IP,
    NETWORK_PROTOCOL_NDP,
    NETWORK_PROTOCOL_SPAWAR_LINK16,
    NETWORK_PROTOCOL_ICMP,
    ROUTING_PROTOCOL_AODV,
    ... ...,

    // WNW MDL
    MI_CES_NM,
    NETWORK_PROTOCOL_CES_WNW_MI,
    MI_MULTICAST_MESH,

    ROUTING_PROTOCOL_NONE // this must be the last one
};
```

b. **NetworkRoutingAdminDistanceType**：network.h 中，给每个路由协议赋予一定的管理距离，指示它的路由优先级，管理距离越小，优先级越高！注意：管理距离大小与在枚举类型声明中的位置无绝对关系。

```
// /**
// ENUM        :: NetworkRoutingAdminDistanceType
// DESCRIPTION :: Administrative distance of different routing protocol
// **/
enum NetworkRoutingAdminDistanceType
```

```
{
    ROUTING_ADMIN_DISTANCE_STATIC = 1,

    // CES
    ROUTING_ADMIN_DISTANCE_EBGPv4_HANDOFF = 111,

    ROUTING_ADMIN_DISTANCE_EBGPv4 = 20,
    ROUTING_ADMIN_DISTANCE_IBGPv4 = 200,
    ROUTING_ADMIN_DISTANCE_BGPv4_LOCAL = 200,
    ROUTING_ADMIN_DISTANCE_OSPFv2 = 110,
    ROUTING_ADMIN_DISTANCE_IGRP = 100,
    ... ...,

    ROUTING_ADMIN_DISTANCE_FSRL = 210,

    // Should always have the highest adminstrative distance
    // (ie, least important).
    ROUTING_ADMIN_DISTANCE_DEFAULT = 255
};
```

c. **NetworkProtocolType**：网络协议类型（相对路由协议而讲，为什么不直接采用上面的NetworkRoutingProtocolType?），定义在 mapping.h中，列出 EXata 所支持的网络协议类型

```
//
// ENUM         :: NetworkProtocolType
// DESCRIPTION  :: Types of various nodes
//
typedef enum
{
    INVALID_NETWORK_TYPE,
    IPV4_ONLY,
    IPV6_ONLY,
    DUAL_IP,
    ATM_NODE,
    GSM_LAYER3,
    CELLULAR,
    NETWORK_VIRTUAL
}
NetworkProtocolType;
```

d. **IpInterfaceInfoType**：定义在network_ip.h，储存IP网络特定接口信息，比如路由协议和调度器类型。

```
//---------------------------------------------------------
// Interface info
//---------------------------------------------------------

// /**
// STRUCT      :: IpInterfaceInfoType
// DESCRIPTION :: Structure for maintaining IP interface informations.  This
//                struct must be allocated by new, not MEM_malloc.  All
//                member variables MUST be initialized in the constructor.
// **/
struct IpInterfaceInfoType
{
    // Constructor.  All member variables MUST be initialized in the
    // constructor.
    IpInterfaceInfoType();

    Scheduler* scheduler;
    Scheduler* inputScheduler;
    NetworkIpBackplaneStatusType backplaneStatus;

    RandomSeed dropSeed;

    D_NodeAddress ipAddress;
    int numHostBits;
    char interfaceName[MAX_STRING_LENGTH];

    // to accomodate all the possible ports..etc
    char* intfNumber;

    RouterFunctionType                      routerFunction;
    NetworkRoutingProtocolType              routingProtocolType;
    void*                                   routingProtocol;
    ... ...;
    // Dynamic address
    AddressState addressState; // current address state
    bool isDhcpEnabled; // Flag to check if DHCP enabled or not.
    Address primaryDnsServer; // allocated by DHCP server
```

```
    Address subnetMask; // allocated by DHCP server
    LinkedList*  listOfSecondaryDNSServer; // allocated by DHCP server
    // Dynamic address statistics
    UInt32 ipNumPktDropDueToInvalidAddressState; // app/network packets
                                                 // dropped due to INVALID
                                                 // address state

};
```

e. **NetworkForwardingTableRow**：定义在network_ip.h，保存Unicast 转发表（Forwarding Table）的一行。

```
// /**
// STRUCT      :: NetworkForwardingTableRow
// DESCRIPTION :: Structure of an entity of forwarding table.
// **/
typedef
struct
{
    NodeAddress destAddress;        // destination address
    NodeAddress destAddressMask;    // subnet destination Mask
    int interfaceIndex;           // index of outgoing interface
    NodeAddress nextHopAddress;     // next hop IP address

    int cost;

    // routing protocol type
    NetworkRoutingProtocolType protocolType;

    // administrative distance for the routing protocol
    NetworkRoutingAdminDistanceType adminDistance;

    BOOL interfaceIsEnabled;
}
NetworkForwardingTableRow;
```

f. **NetworkForwadingTable**：定义在network_ip.h中，描述IP节点中的Forwarding Table，用于 unicast。中间包括指向首行的指针。

```
// /**
```

```
// STRUCT      :: NetworkForwardingTable
// DESCRIPTION :: Structure of forwarding table.
// **/
typedef
struct
{
    int size;                       // number of entries
    int allocatedSize;
    int numStaticRoutes; // number of static routes in routing table
    NetworkForwardingTableRow *row;  // allocation in Init function in Ip
}
NetworkForwardingTable;
```

g. **NetworkMulticastForwardingTableRow**：定义在 network_ip.h 中，保存组播转发表的一行记录。注意到输出接口由一个链表表示。

```
// /**
// STRUCT      :: NetworkMulticastForwardingTableRow
// DESCRIPTION :: Structure of an entity of multicast forwarding table.
// **/
typedef
struct
{
    NodeAddress sourceAddress;
    NodeAddress sourceAddressMask;        // Not used
    NodeAddress multicastGroupAddress;
    LinkedList *outInterfaceList;
} NetworkMulticastForwardingTableRow;
```

h. **NetworkMulticastForwardingTable**：定义在network_ip.h中，保存组播转发表。内部包含一个组播转发行的数组，指向首行记录。

```
// /**
// STRUCT      :: NetworkMulticastForwardingTable
// DESCRIPTION :: Structure of multicast forwarding table
// **/
typedef
struct
{
```

```
    int size;    // number of entries
    int allocatedSize;
    NetworkMulticastForwardingTableRow *row;    // first row
} NetworkMulticastForwardingTable;
```

i. **NetworkDataIP**：定义在network_ip.h，IP 协议的主数据结构，保存网络层 IP 协议的主要信息，包括转发表和组播转发表指针等。

```
// /**
// STRUCT      :: NetworkDataIp;
// DESCRIPTION :: Main structure of network layer.
// **/
struct NetworkDataIp
{
    unsigned short              packetIdCounter;  // Used for identifying
    // datagram
    NetworkForwardingTable      forwardTable;

    BOOL                        checkMessagePeekFunction;
    BOOL                        checkMacAckHandler;
    int maxPacketLength;

    // added for Gateway
    BOOL gatewayConfigured;
    NodeAddress defaultGatewayId;
    ... ...;
     IpInterfaceInfoType* interfaceInfo[MAX_NUM_INTERFACES];
     ... ...;
     BOOL                        ipForwardingEnabled;

    NetworkIpStatsType stats;

    STAT_NetStatistics* newStats;

    LinkedList                          *multicastGroupList;
    NetworkMulticastForwardingTable     multicastForwardingTable;
    ... ...;
    NetworkIpFilter *filters;
    int numFilters;
```

```
32
33      // Dynamic address
34      LinkedList* addressChangedHandlerList;
35  };
```

j. NetworkData：定义在network.h中，保存节点上运行的网络协议类型的数据结构，即网络层的主数据结构，其中包括一个指向 NetworkDataIp 的指针。

```
1   // /**
2   // STRUCT      :: NetworkData
3   // DESCRIPTION :: Main data structure of network layer
4   // **/
5   struct NetworkData
6   {
7       NetworkDataIp* networkVar;  // IP state
8
9       NetworkProtocolType networkProtocol;
10  #ifdef CELLULAR_LIB
11      struct struct_layer3_gsm_str *gsmLayer3Var;
12  #endif
13      struct struct_cellular_layer3_str *cellularLayer3Var;
14
15      BOOL networkStats;  // TRUE if network statistics are collected
16
17      //It is true if ARP is enabled
18      BOOL isArpEnable;
19      //It is true if RARP is enabled
20      BOOL isRarpEnable;
21      struct address_resolution_module *arModule;
22
23      BOOL useNetworkCesQosDiffServ;
24
25  #ifdef ADDON_NGCNMS
26      NetworkResetFunctionList* resetFunctionList;
27  #endif
28
29      struct PKIData* pkiData;
30
31  };
```

k. 在节点结构中包含有各层协议的主数据结构，包括网络层的，NetworkData：

```
// /**
// STRUCT :: Node
// DESCRIPTION ::
//  This struct includes all the information for a particular node.
//  State information for each layer can be accessed from this structure.
// **/

struct Node {
    // Information about other nodes in the same partition.
    Node      *prevNodeData;
    Node      *nextNodeData;

    //! nodeIndex will store a value from 0 to (the number of nodes - 1);
    //! each node has a unique nodeIndex (even across multiple partitions).
    //! A node keeps the same nodeIndex even if it becomes handled by
    //! another partition.  nodeIndex should not be used by the protocol
    //! code at any layer.
    unsigned    nodeIndex;

    NodeAddress nodeId;    //!< the user-specified node identifier
    char*       hostname;  //!< hostname (Default: "hostN" where N is nodeId).

    Int32       globalSeed;
    Int32       numNodes;  //!< number of nodes in the simulation
    ... ...
     // Layer-specific information for the  node.
    PropChannel*    propChannel;
    PropData*       propData;
    PhyData**       phyData;            // phy layer
    MacData**       macData;            // MAC layer
    MacSwitch*      switchData;         // MAC switch

    NetworkData     networkData;        // network layer
    TransportData   transportData;      // transport layer
    AppData         appData;            // application layer
    TraceData*      traceData;          // tracing
    SchedulerInfo*  schedulerInfo;      // Pointer to the info struct for the
```

```
38                                          // scheduler to be used with this node
39      UserData*       userData;           // User Data
40      void* globalData[GlobalData_Count]; // Global Data
41
42      int             numAtmInterfaces;   // Number of atm interfaces
43      AtmLayer2Data** atmLayer2Data;      // ATM LAYER2
44      AdaptationData  adaptationData;     // ADAPTATION Layer
45      ... ...
46      };
```

1.4 网络层 API 与层间通信

a. **传输层到网络层通信**：

- NetworkIpReceivePacketFromTransportLayer: 传输层协议调用此 API 来发包给 IP 协议，定义在network_ip.cpp中。

b. **网络层到传输层**：

- SendToUdp：发一个包给UDP
- SendToTcp：发一个包给 TCP
- SendToRsvp：发一个包给 RSVP

c. **网络层到 MAC 层通信**：根据不同情形有多个 API 可用，它们实现在 network_ip.cpp / mac.cpp。

- NetworkIpSendRawMessage：添加 IP 头给 raw message，然后调用 RoutePacketAndSendToMac 添加路由信息给该包。
- NetworkIpSendRawMessageWithDelay：添加 IP 包头，并调用 RoutePacketAndSendToMac，在一定延迟后，给包添加路由信息。
- NetworkIpSendRawMessageToMacLayer：添加 IP 包头并发送到 MAC 层；
- NetworkIpSendRawMessageToMacLayerWithDelay：添加 IP 包头，一段延迟后发给 MAC。
- NetworkIpSendPacketToMacLayer：发送 IP 包到 MAC 层；
- NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute：添加新的 source route 给 IP 包，然后发送到 MAC；
- MAC_NetworkLayerHasPacketToSend：网络层通知 MAC 层有包发送。

d. **MAC 层到网络层通信**：实现在 mac.cpp。

- NetworkIpOutputQueueIsEmpty
- NetworkIpOutputQueueDequeuePacket
- NetworkIpOutputQueueTopPacket
- NetworkIpOutputQueueDequeuePacketForAPriority
- NETWORK_ReceivePacketFromMacLayer：用于 MAC 层向网络层发送一个输入的包。
- NetworkIpReceiveMacAck：
- NetwokIpNotificatioinOfPacketDrop: 通知上层协议，当 MAC 层产生丢包时。

e. 网络层的 Utility API：定义在network_ip.h

- NetworkIpSetPromiscuousMessagePeekFunction
- NetworkIpSetMacLayerAckHandler:
- NetworkIpSetRouterFunction
- NetworkIpGetRouterFunction
- NetworkIpGetInterfaceAddress

2. 添加一个单播路由协议

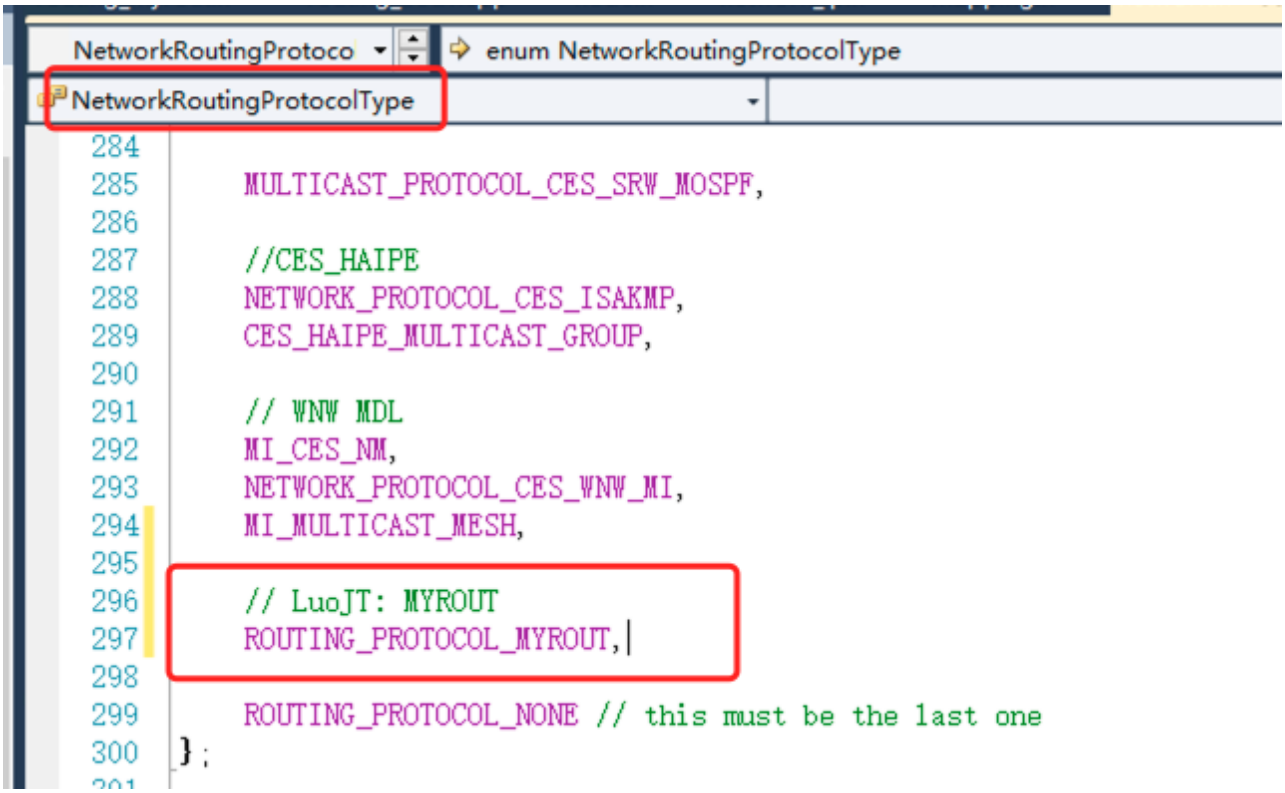
参考的例子是 AODV，定义在libraries/wireless/routing_aodv.h 和 routing_aodv.cpp 中。

2.1 创建文件

在user_models/src/ 下添加 routing_myroutrouting_myrout.h / .cpp。

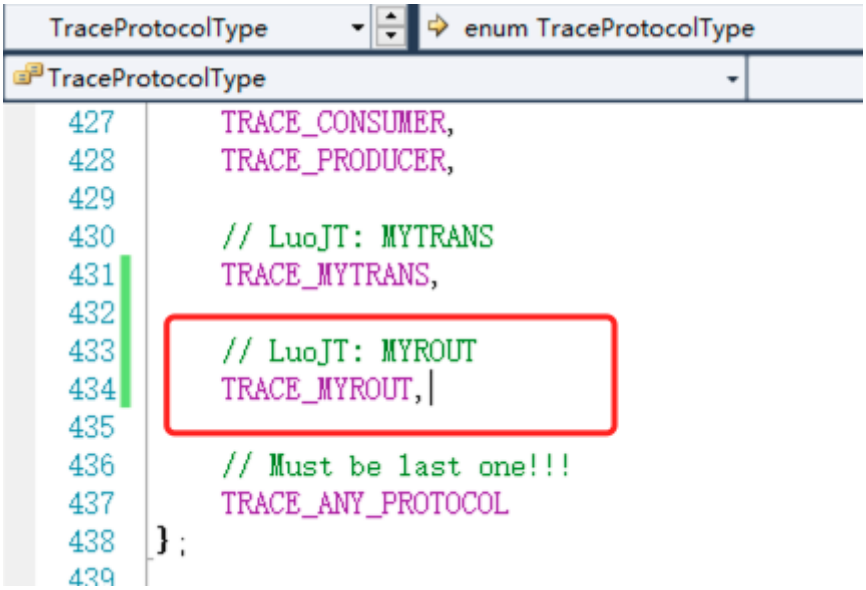
2.2 添加到路由协议列表中

每个节点维护一个路由协议的列表。当网络层添加了一个新的单播路由协议，就需要在Network Layer层协议列表中增加该协议。因此，需要将新协议的标识添加到network.h 文件中的 NetworkingRoutingProtocolType 枚举类型中去。注意：添加到尾部。



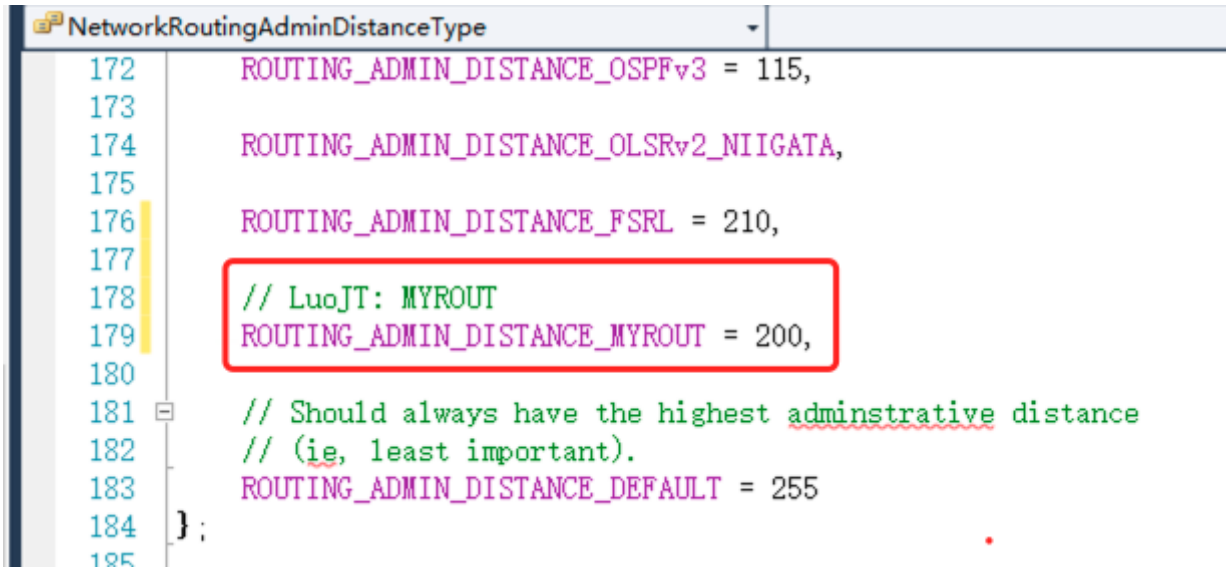
2.3 添加到 Trace 协议列表中

为进行协议追踪方便，把新协议添加到追踪协议列表添加到 TraceProtocolType 中，该列表定义在 include/trace.h中。注意：添加到列表的尾部。



2.4 设定 Admin Distance

Admin Distance 决定一个路由协议的优先级，该值越小，优先级越高。定义在NetworkRoutingAdminDistanceType 类型中，位于include/network.h文件中。这里为新协议设定一个较低的优先级，Admin Distance = 200.



2.5 一起编译

为便于后续扩展时随时编译方便，修改src_models/Makefile-common 文件，添加新建的 source 文件routing_myrout.cpp，并在main下运行nmake，尝试编译。随时编译，检查可能的错误。

Makefile-common | routing_myrout.cpp | routing_myrout.h*

```
USER_MODELS_OPTIONS =


USER_MODELS_DIR = ../libraries/user_models
USER_MODELS_SRCDIR = ../libraries/user_models/src

#
# common sources
#
USER_MODELS_SRCS = \
$(USER_MODELS_SRCDIR)/app_myprotocol.cpp \
$(USER_MODELS_SRCDIR)/app_consumer.cpp \
$(USER_MODELS_SRCDIR)/app_producer.cpp \
$(USER_MODELS_SRCDIR)/hnp_list.cpp \
$(USER_MODELS_SRCDIR)/transport_mytrans.cpp \
$(USER_MODELS_SRCDIR)/routing_myrout.cpp


USER_MODELS_INCLUDES = \
-I$(USER_MODELS_SRCDIR)
```

2.6 定义数据结构

每个路由协议都有自己的数据结构，定义在自己的头文件中。常见的包括：

- 协议参数：用于协议初始化时对协议进行配置；
- 协议状态：用于维护协议状态机。
- 统计变量：用于进行协议性能评估；
- 路由表：最重要的数据结构，是节点进行包转发的依据，也是路由协议的输出。

参考 AodvData （定义在 routing_aodv.h），在routing_myrout.h中添加如下数据结构：

```
// /**
// STRUCT          ::    MyroutData
// DESCRIPTION     ::    MYROUT IPv4/IPv6 structure to store all necessary
//                       informations.
// **/
typedef struct struct_network_myrout_str
{
 // set of user configurable parameters
        Int32 netDiameter;
        clocktype nodeTraversalTime;
        clocktype myRouteTimeout;
        Int32 allowedHelloLoss;
```

```
        clocktype activeRouteTimeout;
        Int32 rreqRetries;
        clocktype helloInterval;
        clocktype deletePeriod;
        Int32 rtDeletionConstant;

        RandomSeed  aodvJitterSeed;

// D- Flag set
        BOOL Dflag;

// set of myrout protocol dependent parameters
        MyroutRoutingTable routeTable;
        MyroutRreqSeenTable seenTable;
        MyroutRreqSentTable sent;
        MyroutMessageBuffer msgBuffer;

        Int32 bufferSizeInNumPacket;
        Int32 bufferSizeInByte;
        MyroutBlacklistTable blacklistTable;
        MyroutStats stats;
        BOOL statsCollected;
        BOOL statsPrinted;
        BOOL processHello;
        BOOL processAck;
        BOOL localRepair;
        BOOL findAlternateRtIfNSet;
        BOOL biDirectionalConn;
        Int32 ttlStart;
        Int32 ttlIncrement;
        Int32 ttlMax;
        UInt32 seqNumber;
        UInt32 floodingId;
        clocktype lastBroadcastSent;

// Performance Issue
        MyroutMemPollEntry* freeList;

        BOOL isExpireTimerSet;
        BOOL isDeleteTimerSet;
```

```
53
54  // Point to MYROUT for IPv6 data if the node is dual IP
55          struct_network_myrout_str* aodv6;
56          BOOL isDualIpNode;
57          MyroutInterfaceInfo* iface;
58          Address broadcastAddr;
59   int defaultInterface;
60          Address defaultInterfaceAddr;
61  } MyroutData;
```

注意到其中有多个数据结构尚未定义，比如 MyroutRoutingTable，还有一些常量需要一并定义。同样参考 AODV 的实现方式。

2.7 初始化

2.7.1 确定协议配置格式

每个路由协议可能用到自己特有的配置参数。这些参数将出现在场景配置文件中。通用的配置格式

```
1  [<Identifier>]  <Parameter-name>  [<Index>]  <Parameter-value>
```

- <Identifier>: 可以是节点标识符、子网标识符，或者 IP 地址，应用到该参数。该参数如果有，则放在[...]内；如果没有，则意味着对所有节点有效。
- <Parameter-name>: 参数名字；
- <Index>: 参数应用的实例序号，放在 [...] 内。适用于有多个实例的情况。如果不含此参数，则意味着对所有实例有效。
- <Parameter-value>: 参数值。

例如，AODV 协议的几个配置参数，如下。当然，如果某参数不出现，则该协议会采用相应的默认值，定义在 default.config。

```
1  AODV-ACTIVE-ROUTE-TIMEOUT  400MS
2  AOEV-RREQ-RETRIES  3
3  AODV-LOCAL-REPAIR  YES
```

2.7.2 调用路由协议的初始化方法

路由协议的依次调用过程如下：

i. PARTITION_InitializeNodes：节点初始化，调用网络初始化，实现在partition.cpp。其中，调用

ii. NETWORK_Initialize：网络初始化，定义在network.cpp，其中分别调用IP初始化和路由初始化。

- NetworkIpInit：IP网络初始化，实现在network_ip.cp，其中，调用

- NetworkIpParseAndSetRoutingProtocolType: 从配置文件中读取各接口路由协议名称，并设定各接口路由协议类型；【这里需要补充新的协议类型】

- IpRoutingInit：IP路由初始化，实现在network_ip.cpp.s，其中，调用各接口所配置的路由协议的初始化方法。

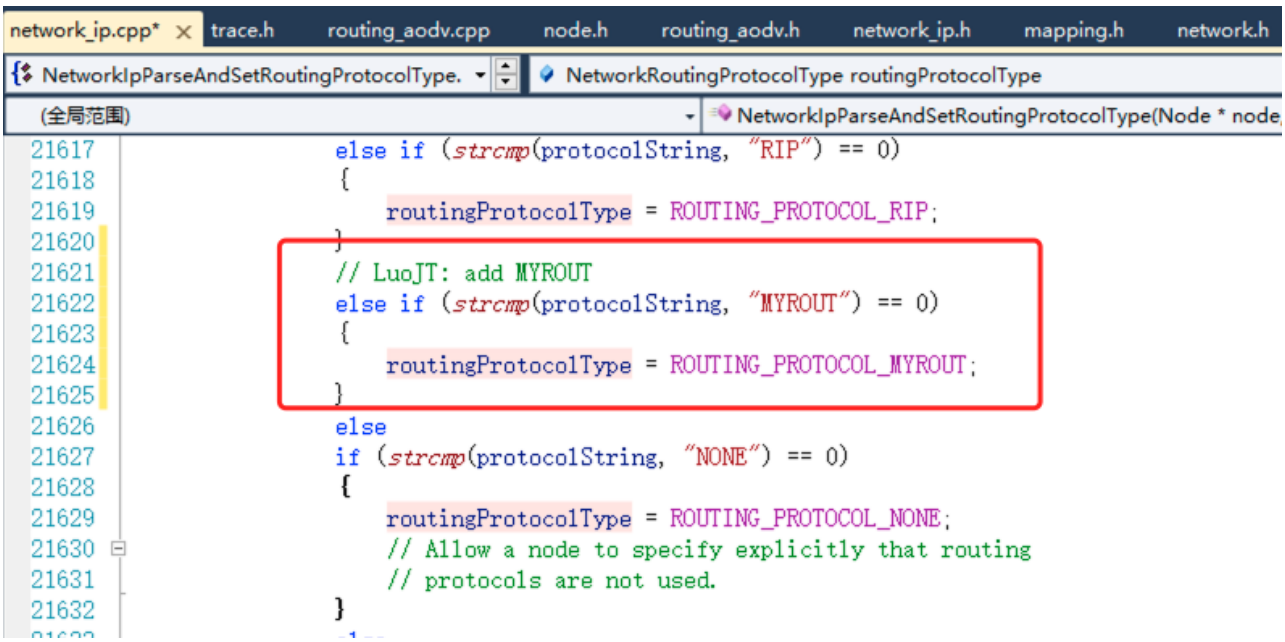
几个重要的方法，如下：

- IO_ReadString：定义在include/fileio.h，从场景配置文件中读取字符串参数。
- NetworkIpGetInterfaceAddress：读取某接口的 IP 地址，定义在network_ip.cpp中。
- NetworkIpAddUnicastRoutingProtocolType：添加一个单播路由协议到一个接口，定义在networkk_ip.cpp中。
- NetworkIpGetRoutingProtocol：返回特定路由协议的数据结构，定义在network_ip.cpp中。注意：如果多个接口运行相同的路由协议，则这些接口共享一个数据结构（一个对象）。
- MyroutInit：新协议需要添加的方法，用于在接口路由初始化时调用。比如 AodvInit 定义在 routing_aodv.cpp 中。它将创建相应协议的数据结构实例（对象），并与相应接口进行关联。
- NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction：用于将多个接口关联同一个路由协议对象。

对于一个新的路由协议，需要完成以下两个步骤：

a. **初始化接口路由协议信息**

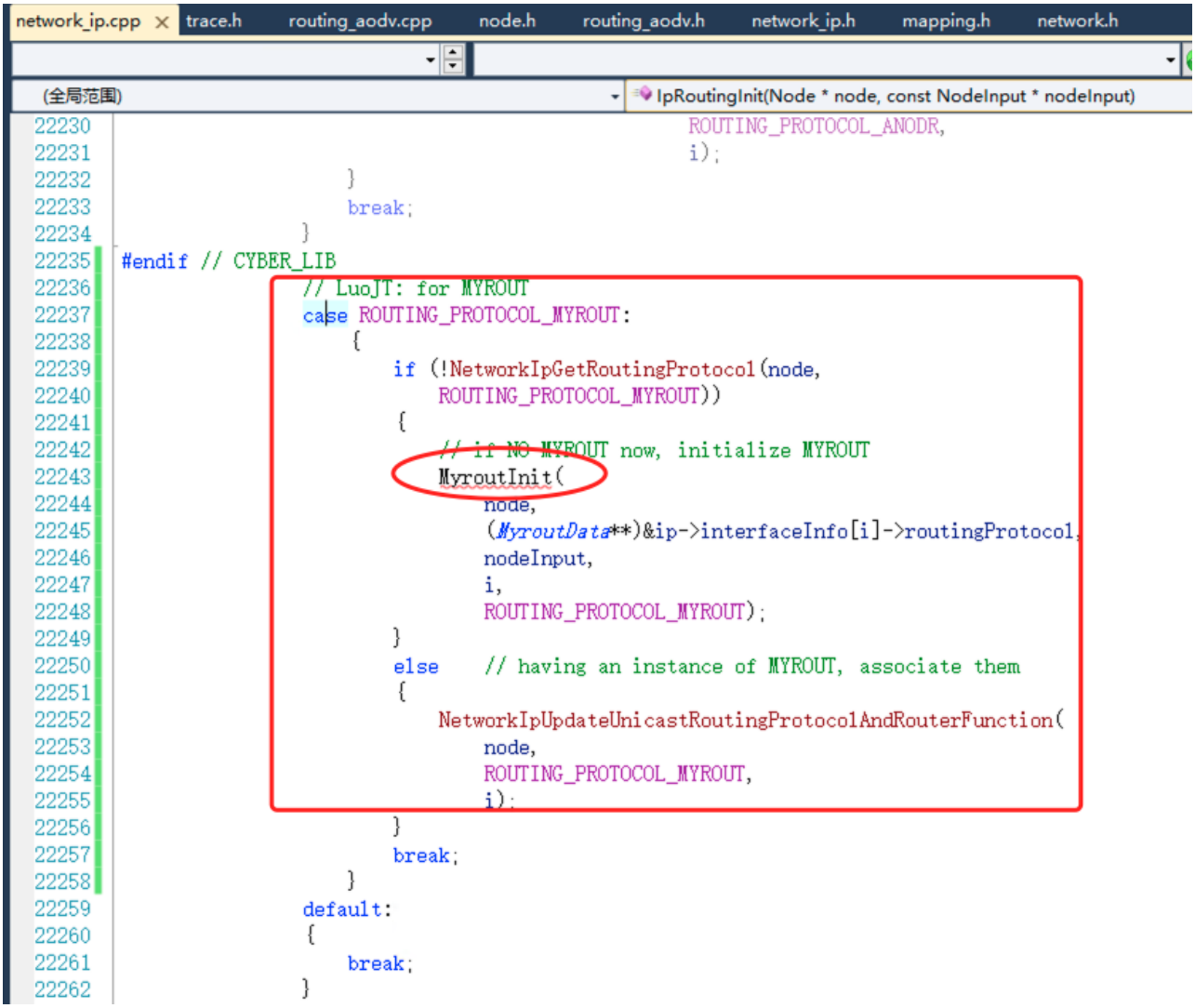
即读取配置文件，将新的路由协议配置参数赋予相应节点及接口。在 NetworkIpParseAndSetRoutingProtocolType 方法中，进行判断，如果配置文件中有新协议的名称，则对对应接口设定相应的路由协议类型。



b. **调用新协议的初始化方法**

在 IpRoutingInit 方法中，当某接口路由协议为新协议 MYROUT 时，则查询当前节点是否存在该协议对象，如果不存在，则调用新协议的初始化方法；否则，则关联已有协议对象。

注意：network_ip.cpp要包含新协议的头文件，否则数据结构无法识别；另外，MyroutInit 方法待实现。



2.7.3 实现初始化

在routing_myrout.cpp和.h 文件中添加 MyroutInit 方法，以及其中用到的一些列方法和常量，完成以下主要功能：

i. **创建新协议实例**（对象）：调用MEM_malloc；

ii. **读取和保存配置参数**，包括

1. 创建并设定全部接口信息：iface 列表，调用MEM_malloc
2. 设定统计和追踪、追踪初始化：MyroutInitTrace
3. 初始化配置参数：MyroutInitializeConfigurableParameters，其中分别调用IO_ReadXXX 读取参数激活状态及设定值，对实例进行初始化。

iii. **初始化路由表**：

1. 清空系列表：HashTable、Expire、Delete、seenTable （用于 RREQ）等；
2. 分配内存池： 调用静态函数 MyroutMemoryChunkAlloc()；

iv. **注册路由器函数，以及 IP 的回调函数**（注意：IPv4 和 IPv6 版本不同）：新的路由协议主要利用 IP 协议完成包的路由，并处理有关事件。为此，路由协议在初始化时将传递一些函数指针给 IP 协议，这些函数称为回调函数（Callback function）。 运行中，当 IP 遇到需要新协议处理的事件，即可直接调用这些函数。

1. **本协议的路由器函数：RouterFunction，在 MyroutInit 中直接调用。**
    a. **NetworkIpSetRouterFunction：**
2. **MAC 层状态信息事件Handler**：主要是针对下层链路失效 Link Failure 时的操作，比如下跳节点的重新选择；IPv4 和 IPv6 分别调用 NetworkIpSetMacLayerStatusEventHandlerFunction、Ipv6SetMacLayerStatusEventHandlerFunction；注意：钩子函数需要单独定义
    a. 涉及 MAC LinkFailure 事件的处理函数，以及系列静态方法的实现.。比如 MyroutSendRouteErrorForLinkFailure 等。完全仿照 AODV 的实现，routing_myrout.cpp 超过 4500 行。这里对所有接口，注册**MACLayerStatusHandler** 和 **RouterFunction** 两个回调函数。注意：IPv4 和 IPv6 有不同版本的实现。代码片段如下：

```
// Check enability of MYTOUR on particular interface and set respective
// MYROUT flag for further use.
for (i = 0; i < node->numberInterfaces; i++)
... ...
// Register router function and IP callbacks
                if (NETWORK_IPV4 == myrout->iface[interfaceIndex].ip_version)
                {
                        // Set the MAC status handler function
                        NetworkIpSetMacLayerStatusEventHandlerFunction(
                                node,
                                &Myrout4MacLayerStatusHandler,
                                interfaceIndex);

                        // Set the router function
                        NetworkIpSetRouterFunction(
                                node,
                                &Myrout4RouterFunction,
                                interfaceIndex);
                }
 ... ...
```

    b.
3.

3. 添加一个组播路由协

4. EXata队列协议

5. EXata调度器