EXata扩展（十）：资源扩散协议 RDP

目标：将Consumer 和 Producer 统一为 RDP 协议的客户端和服务端。

动机：因为二者本身就是服务与被服务的关系，作为一个协议的两段更合理，而且设计更简化；另外，便于进行协议追踪。

1. 命名修改

- 修改应用层协议名称：将 APP_CONSUMER 改为 APP_RDP_CONSUMER ； APP_PRODUCER 改为 APP_RDP_PRODUCER.
- 合并一个 TRACE协议：TRACE_RDP；注释 TRACE_CONSUMER 和 TRACE_PRODUCER，在 trace.h 文件中。
- 函数名尽量保持不变：
- 所有 HNP 改为 RDP；所有 Hnp 改为Rdp；
- 注意：修改.h 文件名时， cpp 文件包含的文件名一并修改；
- 修改 hnp_list.cpp 要修改 user_models/src 下的 Makefile-common
- 合并 ConsumerData 和 ProducerData，统一为 RdpData，在 rdp_common.h 文件中。
- 【修改完毕】

2. 追踪 RDP 协议

2.1 EXata 协议追踪原理

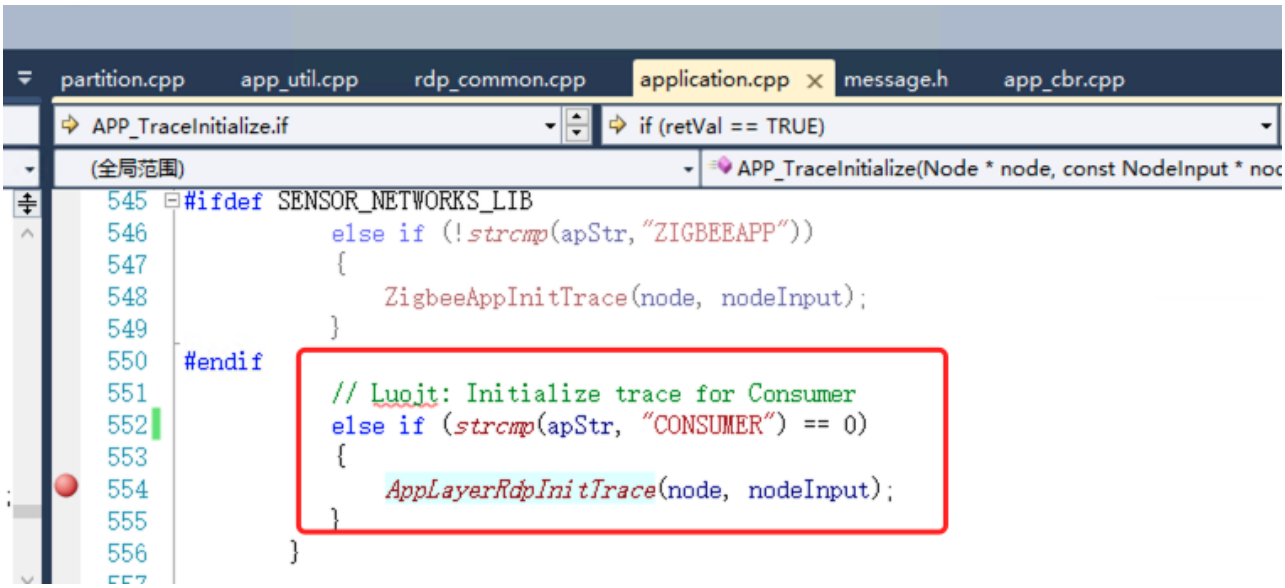
就应用层协议追踪而言，EXata 设计了比较巧妙的机制。

1. **追踪初始化**：每个应用层协议提供一个本协议的**追踪初始化函数**，比如对于 RDP 协议来说，该函数为 AppLayerRdpInitTrace (Node*, const NodeInput*)。该函数被应用层的Trace初始化函数 APP_TraceInitialize 调用，主要完成两个工作：
   a. 读取场景配置文件，确定本协议是否激活追踪。
   b. 注册打印Trace信息函数指针，比如 AppLayerRdpPrintTraceXML。该函数设定实际打印的内容，被通用的Trace打印函数 TRACE_PrintTrace 调用，执行本协议的实际打印动作。
2. **XML Trace 打印方法**：这是每个协议控制本协议追踪记录格式的地方。该函数需要在初始化时注册。对于RDP，其函数命名为：AppLayerRdpPrintTraceXML(Node*, Message *)。
3. **打印 Trace 信息**：在需要打印的地方，直接调用 TRACE_PrintTrace 调用。
4. 出现 Bug：程序意外退出！！！

```
Attempting license checkout (should take less than 2 seconds) ...Loading scenario ex_1.config
Partition 0, Node 1 (285.38, 976.54, 0.00).
Partition 0, Node 4 (1027.94, 1175.82, 0.00).
Initialization completed in 0.103 sec at 2023-02-17 23:30:54.207
Current Sim Time[s] =    0.794842814  Real Time[s] =    0  Completed  1%
Current Sim Time[s] =    0.794894014  Real Time[s] =    0  Completed  2%
CONSUMER: Node 1 is sending a message
abort

F:\ex\ex_1>
```

5. 根源可能是这儿的问题：APP 配置文件 ex_1.app 中的用的协议名称分别为 Consumer 和 Producer 各 1 行，但 Trace 的初始化函数只有一个 AppLayerRdpInitTrace。



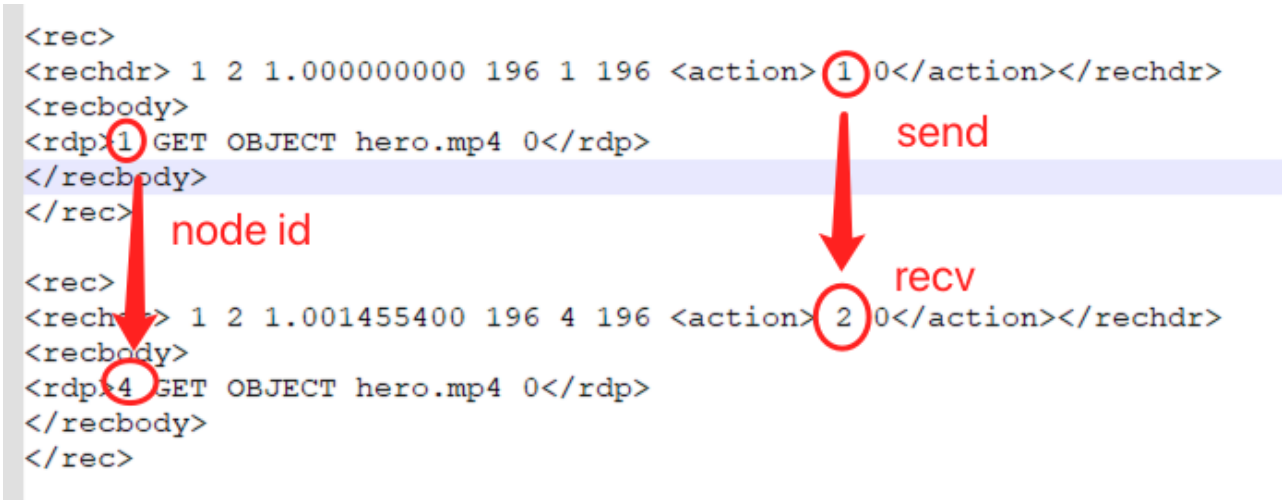
6. 【异常退出解决】是因为打印参数个数不匹配，这里多了一个“%s”。应该是在注释打印时间时没有去掉第一个 %s 所致！

```
26
27  //  TIME_PrintClockInSecond(data->txTime, clockStr);
28
29      sprintf(buf, "<rdp>%s %d %s %s %s %d</rdp> ",
30          node
31          RdpMsgTypeString[data->msgType],
32          RdpResourceTypeString[data->rscType],
33          data->obInfo.name,
34          data->offset
35          );
36      TRACE_WriteToBufferXML(node, buf);
37
38  }
```

int sprintf(char *_Dest,const char *_Format,...)

2.2 RDP 追踪显示优化

1. 此时能够正常输出 Trace 文件：



2. 优化trace 显示，直接在 RDP 记录中打印出方向：send or recv；并省略资源类型 Object or service，以后可以在名字上加以区分。目前效果如下：

```
<rec>
<rechdr> 1 2 1.000000000 196 1 196 <action> 1 0</action></rechdr>
<recbody>
<rdp>1 send GET hero.mp4 0</rdp>
</recbody>
</rec>

<rec>
<rechdr> 1 2 1.001455400 196 4 196 <action> 2 0</action></rechdr>
<recbody>
<rdp>4 recv GET hero.mp4 0</rdp>
</recbody>
</rec>
```

节点1 发送 GET 消息，请求 hero.mp4

节点 4 收到了该消息

3. 节点4 发布该内容【注意：这里故意滞后GET 之后发布，producer缓存请求】，并即时响应请求。注意：这里设定hero.mp4共有 10 个chunks，分别用 offset 来指定；

```
<rdp>1 send GET hero.mp4 0</rdp>
</recbody>
</rec>

<rec>
<rechdr> 1 2 1 001455400 196 4 196 <action> 2 0</action></rechdr>
<recbody>
<rdp>4 recv GE  hero.mp4 0</rdp>
</recbody>
</rec>

<rec>
<rechdr> 4 2 2. 000000000 196 4 196 <action> 1 0</action></rechdr>
<recbody>
<rdp>4 send PUB hero.mp4 0</rdp>
</recbody>
</rec>

<rec>
<rechdr> 4 3 2.0 00000000 196 4 196 <action> 1 0</action></rechdr>
<recbody>
<rdp>4 send GET_RSP hero.mp4 0</rdp>
</recbody>
</rec>

<rec>
<rechdr> 4 4 2.000000000 196 4 196 <action> 1 0</action></rechdr>
<recbody>
<rdp>4 send GET_RSP hero.mp4 1</rdp>
</recbody>
</rec>

<rec>
<rechdr> 4 5 2.000000000 196 4 1 6 <action> 1 0</action></rechdr>
<recbody>
<rdp>4 send GET_RSP hero.mp4 2</rdp>
</recbody>
```

offset

4. 补充显示净荷大小：

```
<rec>
<rechdr> 4 12 2.000000000 196 4 196 <action> 1 0</action></rechdr>
<recbody>
<rdp>4 send GET_RSP hero.mp4 9 1024</rdp>
</recbody>
</rec>

<rec>
<rechdr> 4 2 2.001455400 196 1 196 <action> 2 0</action></rechdr>
<recbody>
<rdp>1 recv PUB hero.mp4 0 0</rdp>
</recbody>
</rec>

<rec>
<rechdr> 4 3 2.002729000 196 1 196 <action> 2 0</action></rechdr>
<recbody>
<rdp>1 recv GET_RSP hero.mp4 0 1024</rdp>
</recbody>
</rec>
```

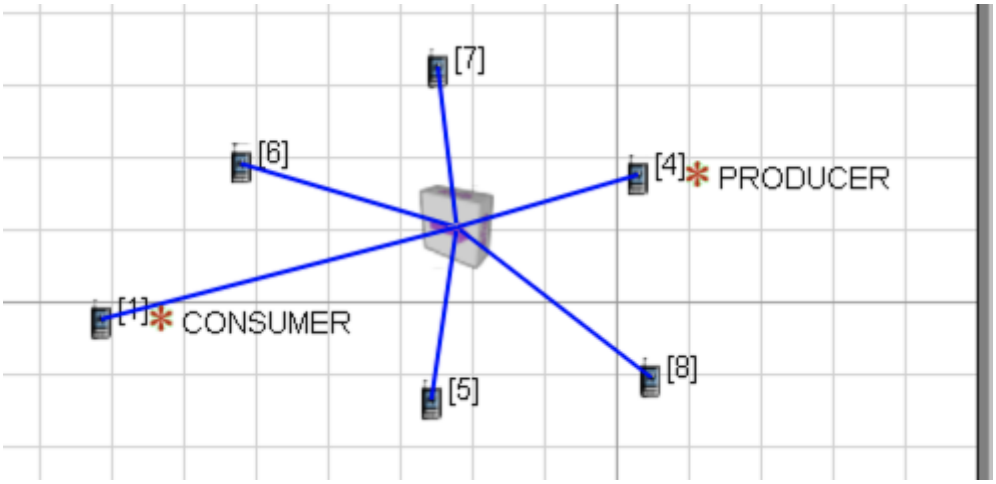
5. 此时，EXata 的 Packet Tracer 也可以追踪到 RDP 协议的包

Index	Type	Mark	Tracing Node	Tracing Protoco	Simulation Time	Originating Nod	Message Sequence Number	Originating Protocol	Action Type
1			1	RDP	1	1	2	RDP	SEND
2			4	RDP	1	1	2	RDP	RECV
3			4	RDP	2	4	2	RDP	SEND
4			4	RDP	2	4	3	RDP	SEND
5			4	RDP	2	4	4	RDP	SEND
6			4	RDP	2	4	5	RDP	SEND
7			4	RDP	2	4	6	RDP	SEND
8			4	RDP	2	4	7	RDP	SEND
9			4	RDP	2	4	8	RDP	SEND
10			4	RDP	2	4	9	RDP	SEND
11			4	RDP	2	4	10	RDP	SEND
12			4	RDP	2	4	11	RDP	SEND
13			4	RDP	2	4	12	RDP	SEND
14			1	RDP	2	4	2	RDP	RECV
15			1	RDP	2	4	3	RDP	RECV
16			1	RDP	2	4	4	RDP	RECV
17			1	RDP	2	4	5	RDP	RECV
18			1	RDP	2	4	6	RDP	RECV
19			1	RDP	2	4	7	RDP	RECV
20			1	RDP	2	4	8	RDP	RECV
21			1	RDP	2	4	9	RDP	RECV
22			1	RDP	2	4	10	RDP	RECV

3. 多跳问题

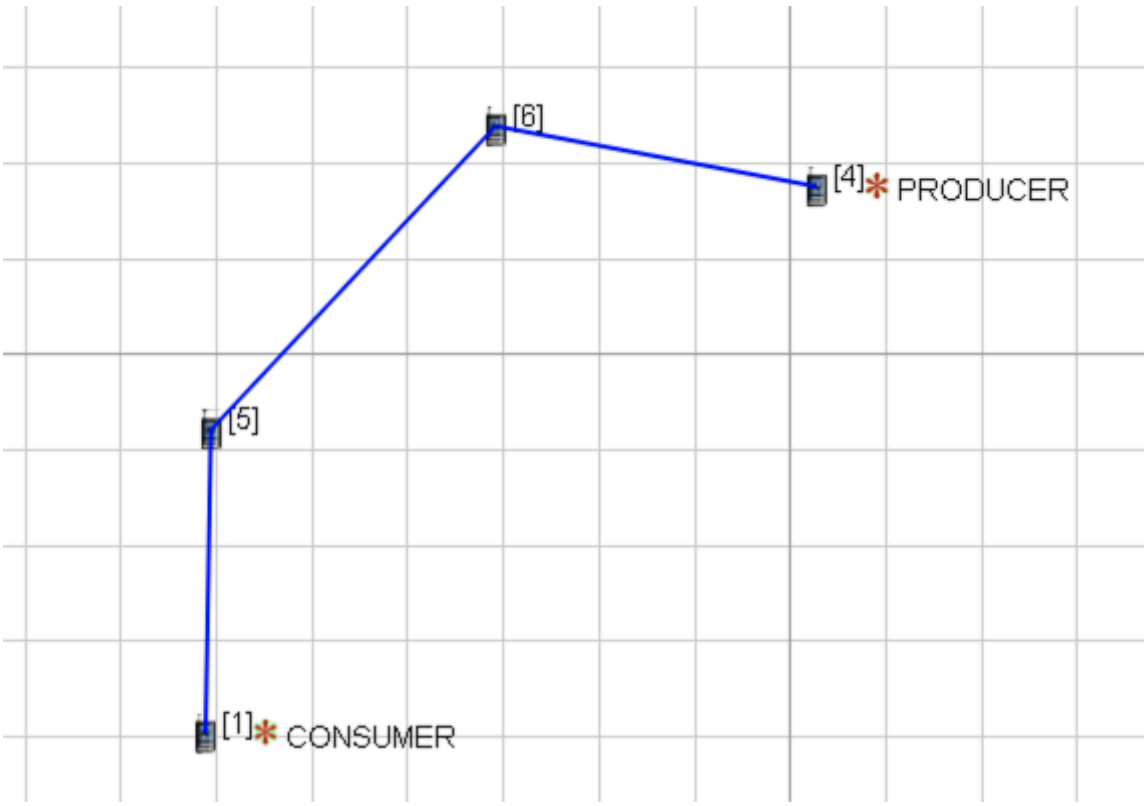
3.1 局域网场景

（1）一对一情形：即一个 Producer 和一个 Consumer。对于LAN的设置，consumer 和 producer 可以正常进行内容请求和接收。



（2）一对多情形：即一个 Producer 和 多个 Consumer。对于

3.2 多跳场景



目前，RDP Consumer和 Producer 的默认对端地址为D类IP 局域网内多播（Multicast）地址：224.0.1.0，从224.0.1.0–224.0.1.255 均为保留地址，不送往广域网。因此，采用默认一个组播地址的 224.0.1.0 为目的地址，无法多跳传输。

采用默认的广播地址（255.255.255.255）也是一样。

因此，对于上图的多跳场景，Consumer 和 Producer 的消息只能传递一跳，无法实现广域网广播，当然，这也是互联网设计在安全方面的考虑。除非大家加入多播组，但这不符合 RDP 共享的设计思路。

目前看来，根本解决途径，必须建立自己的路由才行，也就是借助自己的统一适配层：HNP。

4. 特殊情况

4.1 Producer 请求信息与发布信息不一致

这里是指 Consumer 请求的内容名字与发布的内容名字一样，但其他属性比如 chunksNum，producerName 不一致。

Producer 处理的基本原则：1）producer name 不一致，不处理。2）chunks number 不一致，以 producer 自己发布的为准（显而易见）。

4.2 consumer 请求之前收到响应数据

应该不予处理。即节点如果没有自己发出相应的请求（即请求列表中没有包含），则忽略该应答。未来由 HNP 层进行按照缓存策略进行缓存。

解决方案：

- Consumer 数据结构中添加一个已请求的对象信息列表，保存已请求的对象信息；
- 收到GET_RSP 时，检查自己是否请求；
    - 如果已请求，则收下；
    - 否则，则忽略；即便即将请求。