三层交换机：第一次，根据路由查找出接口，并且学习mac地址，之后的报文都根据mac地址来转发。可以减少广播风暴。vlan报文也可以再三层交换机中去掉。

使用IP的设备A----------三层交换机----------使用IP的设备B
比如A要给B发送数据，已知目的IP，那么A就用子网掩码取得网络地址，判断目的IP是否与自己在同一网段。

如果在同一网段，但不知道转发数据所需的MAC地址，A就发送一个ARP请求，B返回其MAC地址，A用此MAC封装数据包并发送给交换机，交换机起用二层交换模块，查找MAC地址表，将数据包转发到相应的端口。

如果目的IP地址显示不是同一网段的，那么A要实现和B的通讯，在流缓存条目中没有对应MAC地址条目，就将第一个正常数据包发送向一个缺省网关，这个缺省网关一般在操作系统中已经设好，对应第三层路由模块，所以可见对于不是同一子网的数据，最先在MAC表中放的是缺省网关的MAC地址；然后就由三层模块接收到此数据包，查询路由表以确定到达B的路由，将构造一个新的帧头，其中以缺省网关的MAC地址为源MAC地址，以主机B的MAC地址为目的MAC地址。通过一定的识别触发机制，确立主机A与B的MAC地址及转发端口的对应关系，并记录进流缓存条目表，以后的A到B的数据，就直接交由二层交换模块完成。这就通常所说的一次路由多次转发。

表面上看，第三层交换机是第二层交换器与路由器的合二而一，然而这种结合并非简单的物理结合，而是各取所长的逻辑结合。其重要表现是，当某一信息源的第一个数据流进行第三层交换后，其中的路由系统将会产生一个MAC地址与IP地址的映射表，并将该表存储起来，当同一信息源的后续数据流再次进入交换环境时，交换机将根据第一次产生并保存的地址映射表，直接从第二层由源地址传输到目的地址，不再经过第三路由系统处理，从而消除了路由选择时造成的网络延迟，提高了数据包的转发效率，解决了网间传输信息时路由产生的速率瓶颈。所以说，第三层交换机既可完成第二层交换机的端口交换功能，又可完成部分路由器的路由功能。即第三层交换机的交换机方案，实际上是一个能够支持多层次动态集成的解决方案，虽然这种多层次动态集成功能在某些程度上也能由传统路由器和第二层交换机搭载完成，但这种搭载方案与采用三层交换机相比，不仅需要更多的设备配置、占用更大的空间、设计更多的布线和花费更高的成本，而且数据传输性能也要差得多，因为在海量数据传输中，搭载方案中的路由器无法克服路由传输速率瓶颈。

显然，第二层交换机和第三层交换机都是基于端口地址的端到端的交换过程，虽然这种基于MAC地址和IP地址的交换机技术，能够极大地提高各节点之间的数据传输率，但却无法根据端口主机的应用需求来自主确定或动态限制端口的交换过程和数据流量，即缺乏第四层智能应用交换需求。第四层交换机不仅可以完成端到端交换，还能根据端口主机的应用特点，确定或限制它的交换流量。简单地说，第四层交换机是基于传输层数据包的交换过程的，是一类基于TCP/IP协议应用层的用户应用交换需求的新型局域网交换机。第四层交换机支持TCP/UDP第四层以下的所有协议，可识别至少80个字节的数据包包头长度，可根据TCP/UDP端口号来区分数据包的应用类型，从而实现应用层的访问控制和服务质量保证。所以，与其说第四层交换机是硬件网络设备，还不如说它是软件网络管理系统。也就是说，第四层交换机是一类以软件技术为主，以硬件技术为辅的网络管理交换设备。

最后值得指出的是，某些人在不同程度上还存在一些模糊概念，认为所谓第四层交换机实际上就是在第三层交换机上增加了具有通过辨别第四层协议端口的能力，仅在第三层交换机上增加了一些增值软件罢了，因而并非工作在传输层，而是仍然在第三层上进行交换操作，只不过是对第三层交换更加敏感而已，从根本上否定第四层交换的关键技术与作用。我们知道，数据包的第二层IEEE802.1P字段或第三层IPToS字段可以用于区分数据包本身的优先级，我们说第四层交换机基于第四层数据包交换，这是说它可以根据第四层TCP/UDP端口号来分析数据包应用类型，即第四层交换机不仅完全具备第三层交换机的所有交换功能和性能，还能支持第三层交换机不可能拥有的网络流量和服务质量控制的智能型功能。

传统二层交换机，没有学习功能，只能基于mac地址查找。但是对于大型局域网，会带来广播风暴。vlan技术可以，但不同vlan之间的数据转发需要路由器介入。
传统二层交换机功能弱，而专业路由器价格昂贵。
三层交换机，是给二层交换机增加路由功能。第一个报文，走软件路由查找，将这些信息记录到硬件缓存中，后续报文，硬件查询缓存，命中后修改报文的mac、vlan等二层信息，然后发到相应端口，从而实现快速转发。
实际上，三层交换机类似于目前的家庭网关的加强版。家庭网关也有加速功能。
而四层交换机，则是识别报文的80个字节，（最新的一些芯片可以识别到118个字节），然后根据这些信息，实现qos、vpn等高级功能。
真正的三层路由器，需要由路由器的硬件做路由查找，成本要高一些。

9300项目工作量预估

9300项目，按照最初的想法，是软件直接拿3800的平台，稍微做一些改动，适应CMC的业务需求即可。
经过五一节前后和南京的交流，我和严总对于软件方案，有了一些初步的轮廓。基于这个初步轮廓方案，评估如下。
在上述技术方向中，有一些是无法并行的。如下图。其中深色图块，需全人力投入。浅色图块，半人力投入。

	1	2	3	4	5	6	7	8	9	10	11	工作量	人力投入
1. 总体技术方案把控，软件方案验证												8人月	系统工程师、开发经理
2. 对3800业务转发代码的改造												7人月	1~2个骨干员工
3. 对3800接口卡的改造												5人月	一个骨干员工
4. 业务代码移植												14人月	一个骨干员工，两个普通员工
5. BSP及支撑相关												6人月	一个骨干员工，一个普通员工

乐观估计，需要11个自然月，40个人月。且仅任务4和任务5，可以通过增加人力加快进度。

我的建议，如果市场方面的确有强烈三层需求，建议重新做方案选型，明确哪些三层需求是最迫切的，基于3800是否最合适？

如下是对这些问题的技术分析，及工作量预估。
1. 总体技术方案把控，软件方案验证
主要是对软件开发过程中的总体方向，方案做把控。并参与各个技术方向的方案讨论工作。

可以并行做其他事情，但至少要投入一半精力在这个上面。
人力预估：开发经理、系统工程师。
确定软件方案，开发经理和系统工程师，加一名骨干员工全人力配合，1个月。讨论所有软件方案，2个人，0.5的人力。
总体，1个自然月/3个人月确定整体软件方案，5个人月，参与确定各技术方向的详细设计讨论。

2. 对3800业务转发代码的改造
3800是路由器产品，在二层方面功能较弱，比如不支持下行方向修改cos（自测及分析代码，都这样）。
由于二层设备和三层设备的诉求不一致，3800在ACL方面性能较弱，也无法满足我们的需求。
另外一种思路，是使用主控交换芯片来完成二层功能。但经过分析，确认该芯片ACL能力较弱，无法满足需求，且如何和3800现有代码结合，也是问题。因此不具备可行性。
综合这两点，我们需要改造现有3800的转发面代码。
人力预估：
需要1~2个骨干员工，全人力投入。任务：改造3800的转发面代码，使之适应CMC的业务需求。
熟悉代码，2个人，1个月。明确需求，半个月（2人）。写代码，2个月。自测并改bug，2个月。
总体6个自然月，7个人月。

3. 对3800接口卡的改造
3800现有接口卡，包括各种GE电、GE光、语音卡等，最大的作用在于汇聚数据流。完成此目标，一个交换芯片即可搞定。
通过分析3800的系统方案，也的确有这个趋势。就是他们的接口卡，只是主控卡的外延，主控具备完全的控制权。这其中，用到了Local bus或者P1010处理芯片等技术。
但是9300不一样，我们的接口卡有自主能力，即3219、3218芯片。业务代码、控制代码，都可以自主处理。
这种风格上的不一致，导致3800的一个核心能力缺失。即接口卡芯片P1010无法和3219通信。
一种方式，是改接口卡的rosng代码，让它可以ping通3219的linux协议栈。
另一种方式，是改3219的管理方式，砍掉P1010芯片，由3219直接和主控通信。但这种方式，导致管理报文不能送到高优先级队列，业务繁忙时有可能心跳丢失。
无论哪一种方式，都不轻松。
人力预估：一个骨干员工，全人力投入。
任务：主控管理3219。熟悉代码，2个月，讨论方案，1个月，写代码并验证，2个月。预计5个月。
总体，5个自然月。5个人月。

4. 业务代码移植
业务代码移植，指将目前9026上线卡管理相关的CLI命令移植到rosng平台下，然后调试业务需求。
需要注意的是，很多cli命令，其实和业务模块强相关的。比如在9026中，我们需要解析dhcp报文，提取option信息，配置交换芯片。这一过程，在9300中也无可避免。因此，要完成业务代码的移植，需要对RosNG的平台代码有一定了解。
前期田明把RosNG平台的仿真版本跑了起来。但这个版本也只能仿真CLI和数据库调试，无法仿真业务代码。
人力预估：一个骨干员工，两个普通员工，全人力投入。
熟悉代码，2个人，2个月。Demo开发，2个人，1个月。大规模CLI命令一致，3个人，4个月。（基于当初集中管理评估，当时安排2人专门到上海做OLT的命令，武汉安排人在CMC侧对接）
总体，7个自然月，14个人月。

5. BSP及支撑相关
主要完成版本编译、接口卡版本编译、接口卡版本推送、3218和主控之间的5461S phy的管理、主控和3218之间的EPLD管理、3219和P1010之间的phy的管理等事情。
人力预估：1个骨干员工，一个普通员工，可和其他任务并行。
完成版本相关任务，2个月。完成phy、EPLD相关方案设计，2个人，1个月。软硬件联调，1个月。自测及修改故障，1个月。
总体，6个人月。5个自然月。

9300软件方案

刘金成

1　数据转发流程

3218线卡到主控的报文带有CDT头部，也即vlan大于2048。需要将它们转化为业务vlan。类似于1:N的汇聚。

二层汇聚，可以由DX3136交换芯片完成，也可以由XLP208来实现。

1.1　由3136交换芯片实现二层汇聚

3136芯片的关键指标：

Ingress/Egress Policy Rules：　Up to 1K 24B rules

Bridge FDB：　　16K entries

VLANs：　　4K active vlans

从需求的角度分析：

如果上行cos->vlan，下行由交换芯片根据FDB表自动还原，对ACL的规则消耗较小，对FDB表容量要求较高。3136符合我们的要求。

但是如果上行option60->vlan，由于3136的ACL只能识别前24个字节，因此无法满足要求。只能由软件将option60->vlan转化为mac->vlan。这样单线卡可能支持300个CM+CPE，也即只能支持75个并发CM。完全无法满足需求。

从软件开发工作量的角度分析：

一旦选择使用交换芯片来做CDT转换，基本上所有Vlan转换相关的都得在交换芯片上做，工作量主要体现在以下几个方面：

1. Docsis相关vlan转换。比如cos->vlan，mac->vlan，option->vlan等等。当规则越来越多时，还需要考虑规则之间的优先级、规则的动态删除等。
2. 如何配置交换芯片。3800本身是做路由器的，并没有像MAP平台那样配置交换芯片接口框架，有可能需要自己扩展。从9300主控的硬件原理图上，3136和XLP之间的连线比较单一，只有SGMII数据线。
3. 多线卡的影响。多线卡使得情况变得更为复杂，1）软件需要将逻辑接口编号转化为交换芯片的总线编号，2）需要考虑同时收到多个协议报文的并发情景

综合看来，在3136上做二层的汇聚，会破坏3800现有软件结构，工作量大，风险高，不考虑该方案。

1.2 由主控ACL做策略路由

1.2.1 端口映射

在3800路由器中，主控板类型可以区分是否有中心交换芯片C-Switch，接口卡类型可以区分是否有交换芯片L-Switch，是否直连到C-Switch上等多种。为保证数据转发、配置方式的一致性，需要使用端口映射功能将物理端口转化为rosng的全局端口号，从而屏蔽物理走线上的差异屏蔽掉。转发面内部处理流程值看到逻辑上的转发路径，及对应的全局端口号。

RosNG的全局端口号包含slot、port、subport等信息。

Cmts-1/1

Cmts-2/1

1.2.2 9300的接口概念

路由器上的接口可以分为两大类：物理接口和逻辑接口。物理接口是实际存在的接口，如局域网的以太网接口、广域网的POS接口。逻辑接口是需要通过配置来创建，是虚拟接口，如Loopback接口、SuperVLAN接口等。

路由属性端口可配置多个子接口，每个子接口可以封装VLAN ID或者VLAN Range。根据收到报文的VLAN TAG确定其对应的子接口。如报文不带VLAN TAG，则将报文指定到对应的物理接口。确定物理接口之后，通过查相应表项来转发，达到VLAN终结的目的。

为了实现VLAN终结，路由器必须在物理口划分VLAN子接口。每个VLAN子接口可配置VLAN，用于终结报文中的VLAN-ID。由于子接口的数目有限，因此允许一个子接口上面配置多个VLAN，称为VLAN Range。VLAN Range子接口支持8位子接口。

根据VLAN报文结构可以了解到VLAN TAG包含四个字段，分别是TPID，Priority，CFI和VLAN ID。TPID用来标识本数据帧是带有VLAN TAG的数据，长度为16bit。根据VLAN原理可知，在传统的以太网报文基础上，添加4字节的802.1Q的报文头，其中前两个字节为TPID（Tag Protocol Identifier），由TPID标明该报文类型，默认为802.1Q报文，取值为0x8100，常见取值包括0x9100等。但是有些设备厂商不一定完全按照RFC进行实现，所以存在不同厂商设备TPID值不同的情况。所以为了保证兼容性，我司提供了TPID可配置功能，可以跟目前所有厂商进行对接。

1.2.3 Vlan子接口封装单个vlan

1.2.4 Vlan Range子接口封装所有CM的vlan

1.2.5 Vlan tpid配置区分协议报文和数据报文

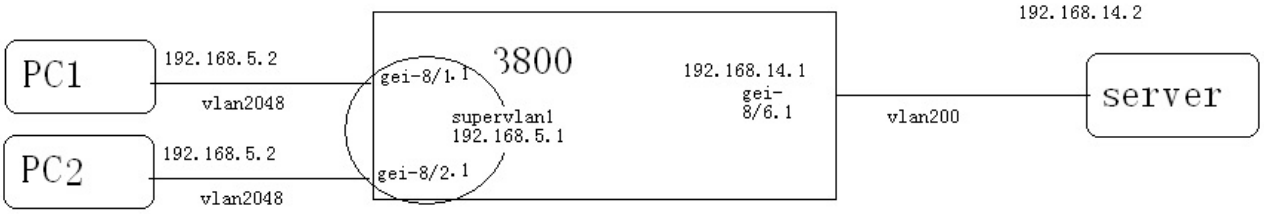
用户口有两个tpid，0x88a8和0x8100。其中0x88a8为数据报文，0x8100为协议报文。所有的协议报文都需要提到cmcController处理，所有的控制报文。

可以做到，协议报文和数据报文从不同的逻辑接口接收。

是否有pvid功能？即收到一个untag报文后，默认修改vlan值。

1.2.6 Super vlan实现多线卡

目的在于实现多个线卡共用一个IP地址，达到节省ip地址的目的，可以使用supervlan实现这一目标。经过验证，该功能可用。下图是模拟验证环境。



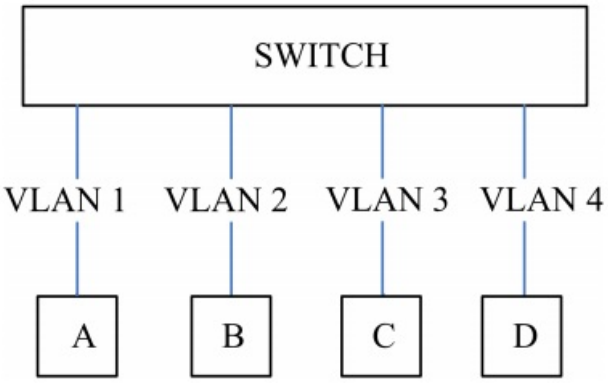
SuperVLAN技术是把多个subVLAN 聚合在一起，这些subVLAN 共同使用一个IP 子网和缺省网关。在SuperVLAN中：

所有的subVLAN 可以灵活地分配SuperVLAN子网中的IP 地址，使用SuperVLAN的缺省网关。

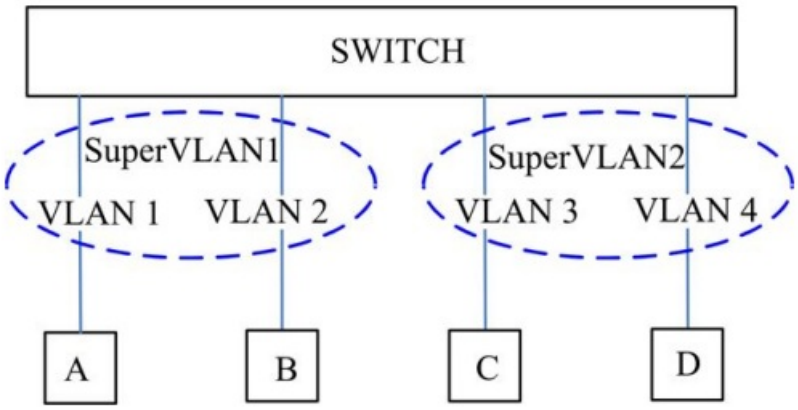
- 每个subVLAN 都是独立的广播域，保证了不同用户间的隔离。
- subVLAN 间的通信通过SuperVLAN进行路由。
- SuperVLAN支持跨板接口绑定和QinQ接口绑定。
- SuperVLAN是由多个接口绑定而成的虚拟接口，可由不同单板的不同VLAN子接口、QinQ子接口、SmartGroup接口或以太网实接口绑定而成。

SuperVLAN由Internet Society的RFC3069提出。在引入VLAN之后，不同VLAN间不能通过二层方式进行通讯，要通讯只能通过三层路由进行转发。由此，不同VLAN间需要配置不同的IP地址子网，出于节省IP地址考虑，引入了SuperVLAN。

普通VLAN的原理如图所示。



设备上连接A、B、C、D之间的端口分属不同的VLAN，因此A、B、C、D必须要配置不同网段的地址，其间通讯通过三层路由转发进行。如下图所示，在引入SuperVLAN之后，将VLAN1、2捆绑到SuperVLAN1，VLAN3、4捆绑到SuperVLAN2。



A、B配置相同网段x.x.x.0/24地址，C、D配置相同网段x.x.y.0/24地址。SuperVLAN1代理A、B之间的ARP，SuperVLAN2代理C、D之间的ARP。因此A、B之间，C、D之间可以通过二层转发进行通讯。当然，A、C这样不同网段地址的主机之间通讯仍需要三层转发。

此外，对于每个SuperVLAN的成员VLAN都可以分配一段IP地址，如果SuperVLAN收到的报文的IP地址和该成员VLAN分配地址不匹配，则丢弃报文，以保障安全性。

1. CM和CPE的网关地址是由provision DHCP服务器自动分配的，而逻辑接口地址是由管理员静态配置的，这中间如何保持一致？不过该问题在所有三层交换机上都存在，应该有解决办法。

1.2.7 路由实现上行出接口选择，即cos->vlan

Cos->vlan，需要使用ipv4-mixed-address-lists做流分类，然后根据策略路由选择出接口。

Option60->vlan需要转化为根据ip做策略路由选择出接口，然后在运行期间动态下发。

和负责ACL模块的SE确认，3800采用算法做ACL规则的查询和生成，效果没有预想那么差，4K条规则可能损失20%的性能。但是，采用算法的后果

是ACL规则配置较慢，大概需要2~3s，且算法计算很占CPU，如果频繁计算，控制面CPU占用率可能占到100%。

因此，ACL只适合于通过命令配置，不适合于运行期间动态删建。

1.2.8 查询ARP表实现下行的vlan和tpid的自动还原

通过实验，已经验证了可以查询FDB表确定出接口，然后自动填充vlan值。

1. 需要确认，能否根据出接口自动填充tpid，能否根据出接口自动填充cos？
2. 如果FDB表查询不到相应的条目，会发生什么事情？

1.2.9 方案1，下行使用qos，实现根据vlan或ip自动还原cos

3800的QoS功能支持报文分类、报文标记、流量监管及整形、拥塞避免、拥塞管理等功能。在报文标记功能中，可以修改dscp、8021p等字段。我们可以使用该功能实现下行的flow2cos功能。

和交换芯片中所说的ACL功能类似，要实现报文标记，需要流分类和报文标记两个模块。其中流分类根据配置的报文特性识别流，然后在报文标记中修改报文内容。

3800的流分类，支持按vlan、802.1p、dscp等字段进行匹配，如果需要匹配ip地址、端口号等信息，则需要根据ACL模块配置的ipv4-access-list作为分类规则，将会收到ACL条目数、ACL性能的限制。

查看用户手册，可以配置规则，根据vlan值修改cos。

能否根据ipv4-mixed-address-lists的结果来做Qos？

1.2.10 方案2，扩展ARP模块，做到cos自动填充

下行cos还原需要考虑几个需求：

1. 一个vlan可能有多个cos值。也即配置上行将多个cos值映射到一个vlan中。
2. 一个mac可能有多个cos值。也就是说一个终端设备，在配置文件中可能会配置到多个cos值。

咱们自己改转发面ARP模块，记录cos值。在回包的时候，不仅填充vlan值，也填充cos值。

目前转发表中ARP条目是这样的：

192.168.5.2 00:00:11 e005.c5ed.68d3 gei-4/2.2 2049 N/A gei-4/2.2

咱们需要在这个ARP表中考虑cos的影响。需要确定下行flow2cos是依据什么字段来区分不同的flow呢？如果是只是单纯的mac，也就是说一个CM/CPE mac只有一个cos，那么下行回包的时候自动根据mac填充cos值。这个很简单，在ARP表中增加一个字段记录cos，上行数据流学习源MAC、源Vlan、源cos，然后下行回包时自动填充这些字段。Tpid的自动填充，是根据逻辑子接口中配置的tpid来决定的。

而如果区分flow的因素不单纯是mac地址，也就是说存在一个CM/CPE的mac有多个cos的情况。这种情况看需要还原的cos值是否和上行cos2vlan值相同

1.2.11 下行组播

3800对组播proxy和snooping的支持情况如何？

下行在组播组洪范时，是否会查询fdb表填充vlan？

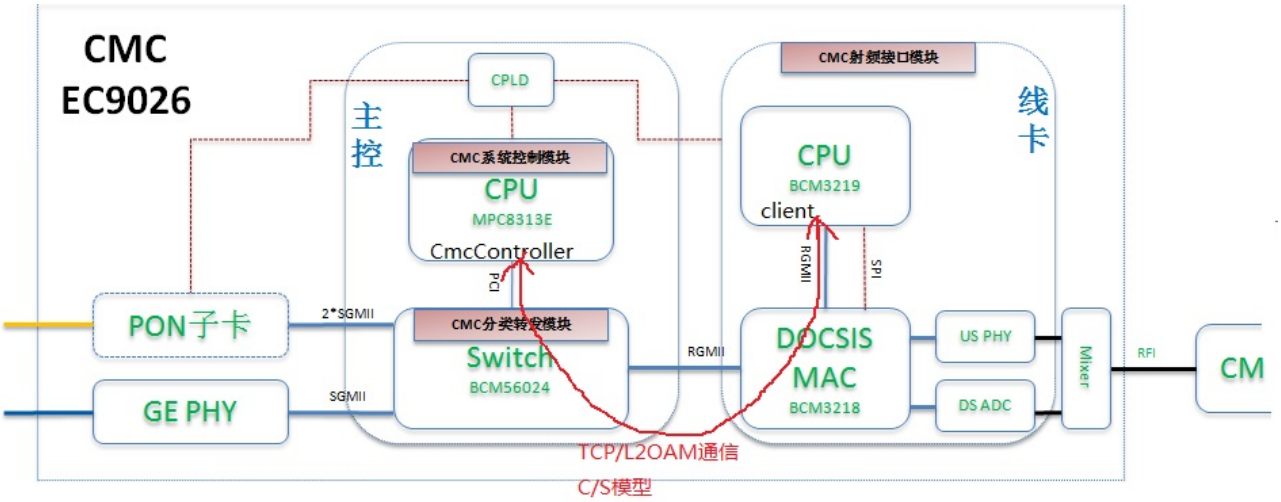
由于9300放在分前端机房，很方便做配置，而且运维人员对于网段划分是非常熟悉的，因此是否可以由运维人员配置静态路由，根据IP地址网段，指定出接口？

2 协议及控制报文转发报文

2.1 CDMM协议报文流程

在9026中，系统控制模块（CmcController）运行在MPC8313上。射频接口模块分为两部分，3218为Docsis MAC，而3219双核CPU上分别运行eCos和Linux操作系统，用于管理DocsisMAC。eCos操作系统实时性高，主要处理Docsis协议相关的调度消息。Linux可操作性好，提供CDMM接口管理射频接口模块。

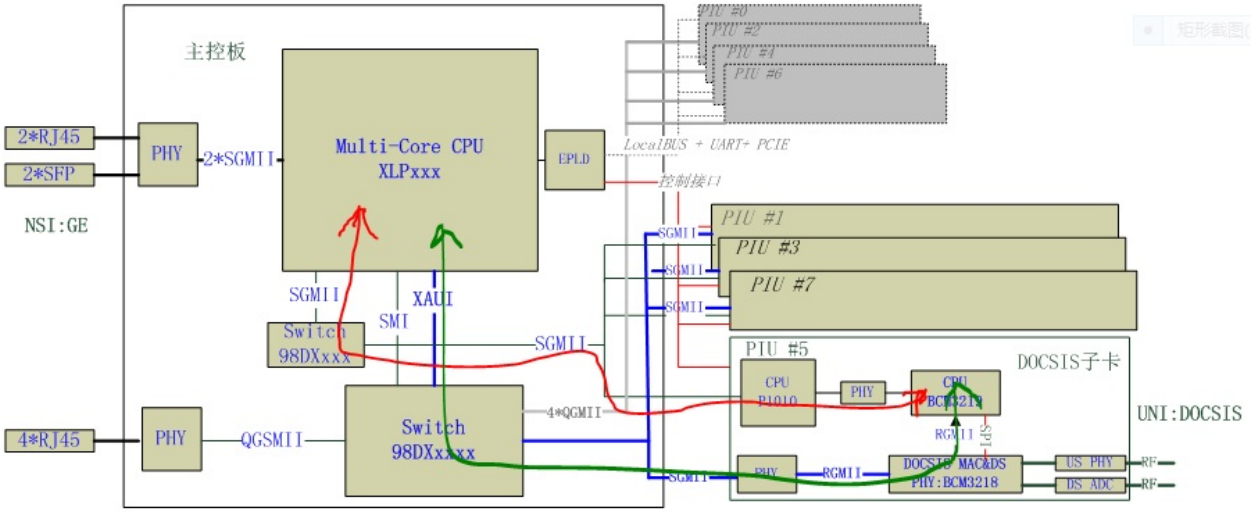
一般地，主控CPU充当Server端，3219 Linux充当Client端。C-S可以使用TCP方式或者L2OAM方式通信。



在9300中，情况有所变化。

首先，需要支持多线卡。在9026中，线卡的IP地址和MAC地址是不会发生变化的，而如果多线卡，每个线卡的IP和MAC需要根据槽位号动态生成，或者使用dhcp协议自动获取。在9026中，所有的CLI命令都没有考虑线卡使用的端口，而9300需要考虑，而且由于支持热插拔，所以端口状态还会可用。

然后，在9300模型中，有控制面和转发面的区分。如下图所示。CDMM在这两个控制平面都可以转发。



2.2 Tulip基础通信框架

在分析9300控制面方案之前，需要了解tuip统一支撑平台。该平台是新一代分布式实时应用程序平台。下图是基于tulip平台应用的典型架构。

所谓分布式，实质tuip平台已经屏蔽了不同硬件、不同的CPU、不同的操作系统之间的差异。应用程序在通讯时，不用关心应用时运行在哪个CPU上，只需要了解进程逻辑地址即可。比如接口卡和主控卡之间通讯，虽然物理上他们位于不同的CPU，但是在逻辑上，由于tulip统一支撑，相关应用程序可以像本地程序一样通信。

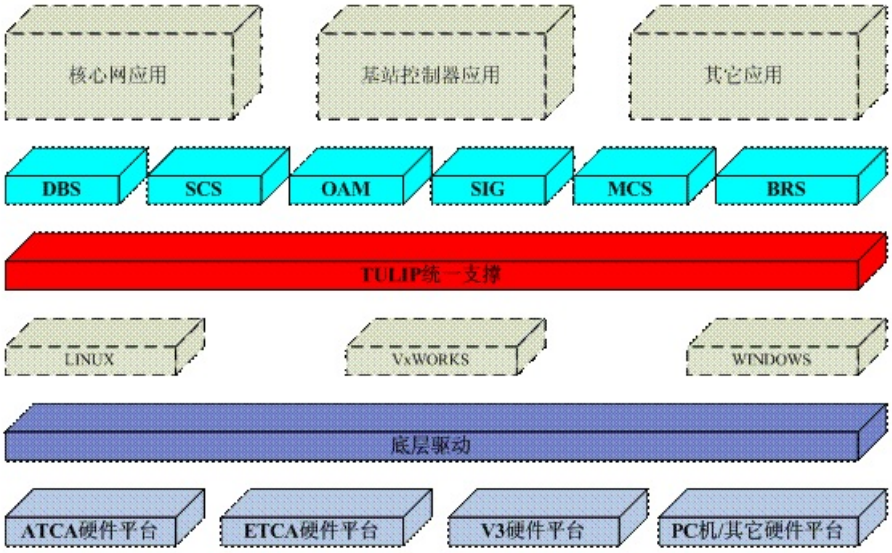


图1-1应用TULIP的软件体系架构典型示例

JOB是承载上层应用功能的载体，接收的输入为消息，内部根据不同的消息进行不同的处理。JOB是统一支撑中的基本调度单位，调度机制为消息驱动。每个消息都有一个消息号作为消息标识，上层应用JOB收到消息后，会根据不同的消息号进行不同的处理。

虽然JOB本身承载在操作系统的线程之上，但是JOB和线程之间存在显著的差异。JOB作为TULIP的基本调度单位，有自己唯一的通信标识（JID），因此任何两个JOB之间都可以通信。

TULIP使用TIPC（Transparent Interprocess Communication）作为通信承载协议，TIPC可以实现透明的进程间通讯。TIPC通讯机制中的连接的双方为JOB，即通讯的两端为JOB。不仅支持跨CPU的TIPC通讯，也支持本CPU内的TIPC通讯。TULIP将TIPC作为一个内核模块进行独立编译、插入内核。

调度线程（Schedule task），用于承载JOB的线程。一级调度，每个调度线程仅承载一个JOB。二级调度，每个调度线程可以承载多个JOB。调度线程的调度是由操作系统实现的，而JOB承载在调度线程下，JOB的调度方式由统一支撑来管理。目前大多使用一级调度，即每个调度线程仅承载一个JOB。

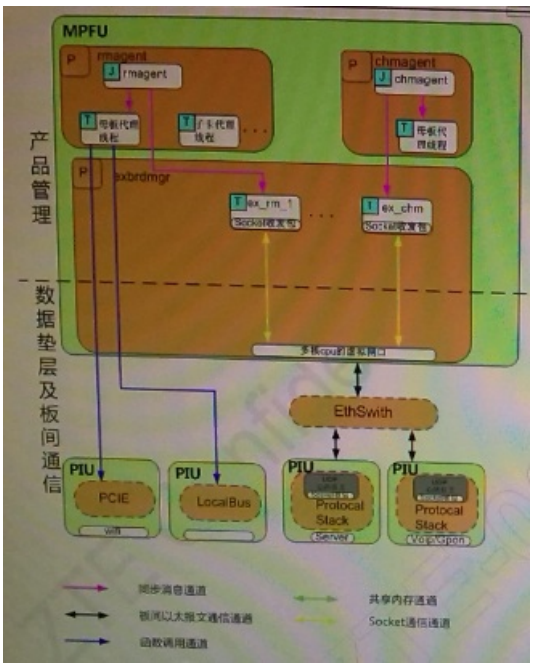
由于JOB使用TPIC作为通讯承载协议，导致它其实不能收发包。当需要收包的时候，还是需要起一个普通的线程来代理它去收发包。

2.3 方案1，通过数据面，OAM模式

3800和固网合作了一块voip的子卡，此种类型的SPIU有CPU，但是不运行CP代码，控制面和媒体面的报文都通过C-Switch到多核CPU。但是和标准的SPIU也有不同，在于它没有local bus和主控卡相连。这款子卡的软件方案已经形成，但是由于最后固网的项目停掉后，这边也没有相关的代码。

由于Docsis接口卡的成本压力，能否不要P1010芯片及内存和flash芯片，参考SPIU来设计，主控卡rmagent使用local bus实现对接口卡的资源管理、机框管理。版本管理和CDMM协议处理则走转发面。

在讨论本方案之前，我们需要先看一下3800产品管理的软件框架。我们只关注其中的控制面转发部分。Ex_rm_1是rmagent的代理线程，rmagent同步调用ex_rm_1代理线程收发包，一对一关系，调用过程中，ex_rm_1处于阻塞状态。而ex_rm_1在控制面交换芯片DX107和XLP相连的总线上创建linux socket监听并收发包。接口卡上，也有相应的收发包代理线程和JOB。但对于应用开发而言，并不会关注到相互通信的两个JOB并不在一个CPU上，因为TULIP会屏蔽掉不同CPU之间的差异。



Voip子卡的设计思路。

从PIU到主控：由PIU主板上CPU 3219的进程从socket发二层以太报文，经过3218 Docsis MAC和C-Switch交换芯片，多核CPU收到该报文，由Marve交换芯片打上DSA_TAG，标识报文的特征。经过NAE投递到相应的VC队列，由cp uCore的软转发识别vlan字段，如果是4094即认为是控制报文，从用户态将报文send到lo口，需要将目的ip为192.的地址替换为127.0.0.1，控制vcpu的exbrdmgr进程中的收包线程绑定到lo接口，即可以收到接收控制报文。

从主控到PIU：在主控的exbrdmgr进程中的子卡代理线程send到lo接口，在转发vcpu起绑定收包线程到lo接口，收到控制报文后，将目的ip替换为某个slot的192网段的ip地址，然后通过NAE发送到C-Switch，L-Switch，由PIU上的收包线程接收到UDP报文，解析相应的数据结构，交给业务线程，完成业务或芯片的相应配置。发送控制报文经过vcpu的好处是BUFFER管理比较清晰，exbrdmgr不需要进行buffer的管理。

在上述的流程中，exbrdmgr和转发vcpu都会绑定收包线程到lo接口上。之所以要将使用该接口，是因为C-Switch和XLP CPU之间有多根总线相连。由于在转发面上就就报文提取，因此相应的物理接口还没有像逻辑接口转换。为避免上层rmagent感知到底层物理接口，所以使用lo接口中转一下，屏蔽底层物理接口的差异。

Docsis子卡的设计思路又有所区别。

由于咱们是新开发一块接口卡，如果不要P1010，可以参考SPIU设计local bus总线，rmagent通过母板代理线程为接口卡做资源管理。而CDMM报文转发路径参考voip子卡，经过lo接口中转。

为实现该方案，还需要对cmcController做JOB改造，将通信相关代码独立成cmcAgent，绑定收发包线程到lo接口上，通过该接口中转、IP伪装等方式和3219通信。而业务处理代码则按照TULIP的风格改造为JOB，方便和数据库、CLI等模块交互。可以保证现有CDMM的通信框架不用做大的调整。

9026将线卡管理和dhcp报文的处理混在一起。而9300中，由于dhcp relay和dhcp snooping等功能使用平台现有的，cmcControlle可以向平台订阅事件，而controller本身尽量专注于对线卡的管理。

存在的问题：

线卡IP地址和MAC地址需要根据槽位号动态生成，防止冲突。需要改动3219的boot代码，在boot阶段识别出本线卡的槽位号，然后自动生成mac地址。不太确定boot能不能这样改。

由于控制面报文的优先级高于媒体面报文，在L-Switch和C-Switch上要保证控制面调度的优先级，另外还要考虑控制面和媒体面的报文在uCore中是否可以区分出来，保证控制面报文进入高优先级的VC队列。从主控到PIU的控制报文通道中，如果从VCPU到NAE，可否保证控制报文的优先级高于数据报文的优先级？如果控制报文的优先级不能保证，是否可能发生丢包的情况？特别是keepalive报文是否可能丢失？

2.4 方案2，通过数据面，TCP模式

参考ICMP、tftp流程实现CDMM交互。

在主控卡上为每个线卡物理接口创建逻辑子接口，封装4094vlan。然后将这些逻辑子接口加入到super vlan中，并为这个supervlan接口分配一个IP地址。在接口卡上，将IP地址绑定到4094vlan的接口上，这样接口卡发送的CDMM报文也封装在4094的vlan中。接口卡直接将报文发送到主控卡，然后被送到supervlan子接口中。控制报文上送，可以利用现有的ARP表机制。当报文发送给本机的话，就上送到相应的逻辑子接口上。Cmcctroller创建一个收包线程绑定到这个逻辑子接口的IP地址上。可以参考icmp、tftp等进程，看下他们怎么实现的。

在9026中，需要主控卡存储线卡的版本，并通过tftp协议传送给线卡。9300中，在主控卡上起tftp服务器，3219充当tftp客户端，采用该方案可以很容易实现版本更换。

相对于方案一，该方案的优势在于不需要经过lo接口中转，controller和3219直接通信。而且有大量的示例代码可以参考，不用从头开发。

但任何事情总有两面性，硬币的另一面，是目前大多数的示例代码都是基于rosng的协议栈，而目前controller代码的tcp socket是起在linux的协议栈的上的。当然rosng协议栈本身也支持建立tcp socket，毕竟需要对controller通信代码做改造。

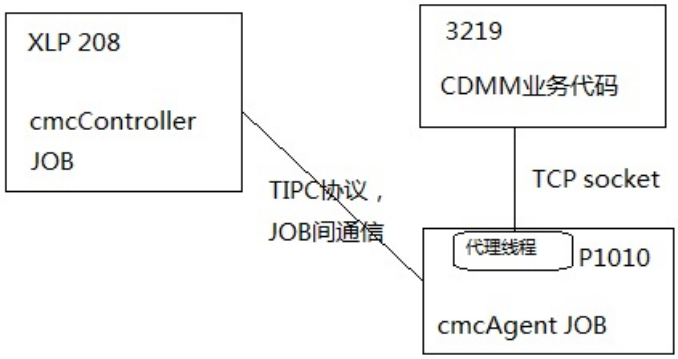
存在的问题和方案一类似，接口卡的IP/MAC需要动态生成，以及如何保证控制报文的优先级。

2.5 方案3，通过控制面，Cmccontroller部署在主控上

2800路由器接口卡并不存储版本文件，估计仅有boot。当接口卡上电后，P1010启动boot，通过dhcp协议向主控要地址，然后再通过网络方式向主控要小系统，linux启动后，再向主控要版本。主控根据配置的xml文件确定需要给接口卡哪些版本文件。之后这些应用程序在接口卡侧启动。由于应用程序本身基于TULIP，JOB之间采用TIPC协议通信，相互通信时不用关心程序运行在哪个CPU上。

本方案的核心思想是，将controller中通信代码移到P1010上，也即在P1010上将TCP的socket终结掉。然后业务代码仍然保留在主控侧。对于需要和CM通信情况，比如查询CM的SNR，由于每个逻辑子接口都有IP地址，因此完全可以将snmp请求通过这个IP地址发给CM。而CM的回包，由于目的MAC是逻辑接口的物理地址，因此也不需要设定特殊的提包规则。

CDMM消息通过TCP socket发给P1010时，由代理线程cmcAgent终结掉，按照固定的消息格式，读取载荷，使用TIPC消息，JOB间通信发送给主控侧cmcController JOB。反之类似，cmcController将要发给3219的载荷封装按照TIPC的方式封装在JOB间通信的消息中，由cmcAgent JOB解析后通过其收包代理线程发送给3219处理。



关于如何给3219换版本，也可以复用现有机制。即将3219的版本文件假装为应用程序，cmcController JOB和cmcAgent JOB约定好后，在主控起FTP server，在接口卡起FTP client，把版本文件取到接口卡内存中。然后利用现有3219版本升级流程，在P1010上起tftp server，由3219将版本拉到其flash中，完成版本升级。

当涉及到多线卡的管理时，首先要指定逻辑接口，比如gei-2/1.1，然后由controller将逻辑接口转化为槽位号，进而可以构造出相应接口卡上cmcAgent的JID，再通过TIPC将CDMM消息发到对应的cmcAgent JOB上，再通过它的代理线程，完成和3219的通信。

由于目前3800没有仿真环境，版本编译、升级，系统启动需要花费大概20分钟的时间，效率相当低下。CmcController本身有命令行代码，希望可以有一套机制，可以复用这些代码。大概的思路是，在命令行中进入cmc模式。在该模式下，所有的输入都会转给Cmccontroller JOB做实际处理。

优点：

1. CDMM消息不经过转发面，不存在优先级问题。可以保证在高负载的情况下，CDMM消息也不会丢失。
2. 多线卡的问题由现有3800机制解决，不需要controller处理多线卡的问题。
3. 3219上的CDMM通信框架不用作任何调整
4. 每个接口卡IP、mac地址需要动态生成的问题，也不存在。
5. 存在利用controller现有cli代码的可能性。

主控和接口卡之间的通讯方式：

主控同步数据库改动给接口卡，接口卡订阅数据库的变动。

也可以接口卡和主控卡之间采用TIPC通信。

2.6 方案4，通过控制面，Cmccontroller部署在P1010上

将controller部署在P1010上，接口卡处理CDMM消息，主控卡和接口卡中间继续使用原有协议通讯，并通过rosng的私有协议，将数据存到主控的数据库中。当主控卡需要主动获取信息时，通过子卡代理线程，将消息发送到P1010上，由P1010将消息转换为CDMM消息后发送给3219。3219回应后，再存入到主控的数据库中。

缺点：

1. 由于在P1010上需要将CDMM消息转化为私有格式，因此预计工作量会稍大。

3 BSP及支撑相关

3.1 多核的分工

目前9300方案中，线卡不占用XLP的物理核。但是9300演进方案中，线卡需要占用XLP的物理核，至少需要四个。

1. 我们需要了解目前在3800产品中，XLP208的8个逻辑核是如何分工的，有没有可能空闲一些核出来？
2. 3800的多核开发，是将某一个物理核绑定为某个功能，还是说需要由应用程序做多核控制？

3800使用XLP208芯片，一共有8个核，其中只有一个用作控制面，其他7个核全部用作转发面。使用的核越多，性能越强劲。正常使用时，大约1Mpps。如果不经过查路由表等步骤，甚至能够有4Mpps的转发性能。

如果使用XLP104，基本上性能就折半了。

在高端的如T8000，有专门的NP芯片负责报文转发，采用微码编程。3800上的NP核上执行的是C语言的代码，经过优化，提升性能。可以理解为软件转发。

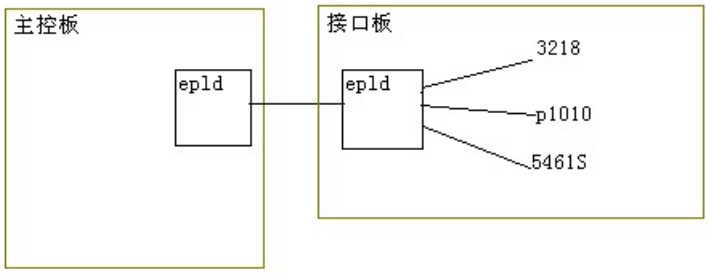
多核之间的协调，采用要令牌方式。

基本上，转发面的7个核，咱们都不用太关心。

3.2 3218和主控之间的phy5461S

3218侧使用RGMII接口，但是主控侧使用SGMII接口。

3.3 主控板通过EPLD配置3218



9300的docsis接口卡，相对与现有接口卡，需要增加通过epld控制3218的，属于新增功能。主控板通过片选寄存器设置接口板的epld通道，选择待配置的器件。然后再和该器件收发数据。

3.4 P1010和3219之间的TCP通信

P1010和linux之间有通信。

4 命令行移植

将9026上的命令行移植到9300上，需要按照Rosng的命令风格重新开发。

5 在模拟环境下开发

希望能够将控制面和转发面的开发工作分离。

3800用户口连9026的GE上联口，56024双向透传。

5.1 3800不参与线卡管理

线卡工作在BRCM模式，cmcController放在3800的上联口的PC上。

5.2 3800参与线卡管理

线卡工作在Docsis模式下，将CmcController移植到主控上。

5.3 仿真环境搭建

南京数通低端路由器那边rosng产品代码没有仿真调试环境。主要是低端路由器产品从高端划分出来的时候，相应的仿真环境维护人员没有划分过来，时间久了，仿真环境也就不再可用。 由于我们做新产品开发，有大量命令行需要调试，后续将会很麻烦。一个是咱们单板不够用，另外一个是升级版本太慢。只要涉及主控板开发，换一次版本差不多要20分钟。

上海OLT也在使用rosng平台，和他们了解到，如果仅仅调试命令行，就直接用NG的发布版本即可，要调试线卡，问下来貌似只能用产品版本。对应平台版本为ROSng V2.00.10。

分析本文第2章仿真环境适用情况：

方案1，通过数据面，OAM模式，控制报文需要经过vcpu控制核、lo接口、cmccontroller JOB等，在仿真环境下难以模拟。

方案2，通过数据面，TCP模式，需要在逻辑子接口上使用rosng的socket和3219通信。因此需要看下rosng平台仿真版本是否支持对逻辑子接口、TCP应用的仿真。

方案3、4，只要能够仿真cli和数据库即可，然后再开发cmccontroller JOB和cmcAgent JOB，由cmcAgent JOB的代理线程和3219通信。因为代理收发包线程直接和linux socket打交道，不依赖于rosng平台的仿真。至于cmcAgent在P1010上运行，对仿真的影响较小。因为仿真环境基于TULIP，可以屏蔽不同CPU之间的差异。当然一些必要的桩函数，还是必不可少的，比如槽位号获取。

综合看来，方案3是最容易使用仿真环境开发的。

6 9300学习

6.1 策略路由

对于路由器收到的报文，首先判断入接口是否绑定了策略路由，如果没有绑定，则按照正常的根据目的地址查找路由表进行转发；如果绑定了策略路由，则按照route-map的sequence 依次进行处理，具体过程如下。

1. 首先用报文去匹配第一个sequence 中配置的ACL，若匹配失败，则接着匹配下一个sequence中配置的ACL，依次类推；若匹配成功，则判断所

属sequence的属性。

2. 若sequence 的属性为deny，则走正常路由；若sequence 的属性为permit，则根据该sequence中的set 项进行转发。

3. 判断是否存在有效的set ip next-hop项。当有多个set ip next-hop项时，按照配置顺序选择第一个有效的下一跳，若存在，则将报文送往设定的下一跳。

4. 若未设置set ip next-hop或不存在有效的set ip next-hop，则需要进一步查看是否存在有效的出接口（该接口存在且状态为UP）。当有多个set interface 项时，按照设置顺序选择第一个有效的出接口。若出接口存在，则报文从该接口直接发出，否则走正常路由。

5. 走正常路由时，若在转发表中查到相应的路由，则按此路由进行报文转发，否则若系统未设置默认路由，则将报文丢弃。策略路由的下一跳为非直连IP 地址时，仍然可以生效，只要能够通过查找本地路由表决定流量的下一跳。

ZXR10(config-route-map)#match ip address *(<access-list-number>在路由映射配置模式下配置match项，**对与访问表匹配的包进行策略路由ACL既可以是标准型的又可以是扩展。型的。**（但经过验证，发现这里并不会根据名称检查ACL是否存在，因此无法验证实际验证是否支持扩展型的ACL）。

6.2 接口

9300并没有严格的上联口用户口的区别。

进入到接口模式，使用description命令，可以设置接口的属性。

6.3 Wifi接口卡

Wifi接口卡通过和主控卡上的PCIe总线直连，不经过中心交换芯片。但在逻辑上，应该仍然只是表现为主控卡的一个端口。

Wifi接口卡使用Athoros平台，是否有可能在接口卡上运行CSP平台，做实际的wifi业务处理？