第二章　链路层（**link layer**）

2.1.　介绍

从第一章图 1.6 中，可以了解到 TCP/IP 协议族中链路层的作用主要有 3 个：

1. 为 IP 模块发送和接收 IP 数据帧；
2. 为 ARP 模块发送请求和接收应答；
3. 为安全准入模块发送和接收控制信息。

TCP/IP 协议族中支持种类繁多的链路层协议，取决于网络硬件类型，常见的硬件网络类型有：

1. *Ethernet*（以太网），最常见的局域网类型，包括光介质（*Fiber*）和铜介质（*Copper*）两种物理类型，介质也就是之前所说的媒介 Medium 或者 Media；
2. *WLAN*（*Wireless LAN*），无线局域网，从原理上来说算是以太网的无线版本，其传输介质是常见的工作在公用频段的电磁波 Radio；
3. 拨号专线，如电话线（铜介质）和 3G（电磁波介质，工作频段需要靠政府牌照）等，运行的链路层协议通常是 *PPP*（*Point-to-Point Protocol*）点到点协议；
4. *PDH*/*SDH*/*SONET*（*Plesiochronous Digital Hierarchy* 伪同步数字序列、*Synchronous Digital Hierarchy* 同步数字序列、*Synchronous Optical Networking* 同步光网络）等租赁专线，通常运行 *PPP* 协议或者 *HDLC*（*High-level Data link Control* 高级数据链路控制）协议；
5. *ATM*（*Asynchronous Transfer Mode* 异步传输模式），本来 ATM 是和 TCP/IP 协议族并行的一种技术，但目前主要是以 TCP/IP 协议族的链路层技术出现；
6. *GPON*/*EPON*/*EPCN*（*Gigabit-capable Passive Optical Network* 千兆无源光网络、*Ethernet PON* 以太网无源光网络、*Ethernet Passive Coax Network* 以太网无源同轴网络）等；
7. 虚拟接口如环回接口 *Loopback*、*GRE* 隧道接口，这是一个特殊的软件接口；
8. 其它新兴网络类型如 *WiMAX*、*WDM*/*DWDM* 等。

这些种类繁多的网络硬件类型应用于不同的链路场合，链路层网络通常是根据链路物理通信范围大小分成：

- *PAN*（*Personal Area Network* 人域网），链路范围通常在 10m 以内，如无线蓝牙就是 PAN 的一种；
- *LAN*（*Local Area Network* 局域网），链路范围通常在 300m 以内，特点是多点接入，这就是最常见的网络链路，如以太网、WLAN 都是局域网链路；
- *MAN*（*Metropolitan Area Network* 城域网），链路范围通常在一个城市以内，SDH、长距以太网等技术可以称为城域网，特点是高速专线；
- *WAN*（*Wide Area Network* 广域网），比城域网范围更大的成为广域网，常见的技术主要是

SDH、WDM 类技术，其特点是点到点专线；

通常而言，局域网的链路范围最适合多点高速通信，价格也比较低廉；广域网适合建立一个跨越全国的大型专线网络，价格高，但速率低；城域网则夹杂在广域网和局域网之间；人域网讲究节能低耗低辐射，所以在研究网络时通常不作为参考对象。互联网是由所有这些网络链路组成，并不存在绝对意义上的孰优孰劣，是否满足当时的产业链的要求是更现实的标准。在网络的历史短河之中，我们看到的只有推陈出新，新人换旧人的场景。在本文中对各种链路层技术的具体物理细节并不作分析，侧重于提炼出链路层在 TCP/IP 协议族中的作用，使用广泛而成熟的局域网、广域网类链路层协议会成为探讨的重点。

在 Richard Stevens 在 1993 年写 TCP/IP 详解时主要使用的网络硬件类型有以太网、令牌环（*Token Ring*）、*FDDI*（*Fiber Distributed Data Interface* 光纤分布数据接口）和 *RS-232* 专线，这些都是当时比较流行的局域网技术。但到了 2010 年，这些网络中也只有以太网幸存下来，并活得很好，并且统治了有线局域网的市场，令牌环、FDDI、RS-232 都已经成为了以太网的刀下之鬼；新兴的无线局域网也有了主人，它就是 WLAN，通俗点叫 Wi-Fi（其实是一个由多家企业组成的 Wireless Fidelity 联盟，旨在推广 WLAN 的标准化和互通），专业一点叫 802.11；拨号网络也从传统的电话线扩展到了 3G，甚至 4G；骨干网广域专线上的技术也是风起云涌，如 PDH、SDH、SONET、WDM、DWDM 等等；网络接入从电话线扩展到了各类线路，如有线电视网络、电力网络。可以说从 1993 年到 2010 年，网络硬件的更新替换、风雨骤变远远比 TCP/IP 协议族变化来得大。但 TCP/IP 以其良好的分层体系，并没有非常大的变化，其本质原因就是分层，网络硬件的变化只需要链路层做相应的变化即可，并不要求网络层或者传输层有什么样的修改，这是 TCP/IP 得以长盛不衰，发展壮大的原因之一。

链路层内有也有细节性的机制来适配网络物理类型和传输层，链路层发展至今，已经可以实现链路层协议的叠加，如 *PPPoE*（*PPP over Ethernet*）、*PPPoEoA*（*PPP over Ethernet over ATM*）、*MAC in MAC*（即 *Ethernet in Ethernet*）等。纵观 TCP/IP 协议族中，网络层和传输层都已经比较成熟，革命性的突破很少，而就属链路层最为风起云涌。但本文从概述中即表明了本文的意图，立足于 TCP/IP 层与层的原理性详解，对于链路层的网络硬件类型、底层实现机制并不作过多介绍，也不一一介绍各网络硬件类型及技术，着重于介绍目前比较流行的以太网、WLAN、PPP、环回接口（Loopback）等协议封装原理，愈来愈重要的安全准入也会进行简单介绍。此外 *MTU*（*Maximum Transmission Unit* 最大传输单元）、最大及最小包长、线速传输也是链路层的一个重点内容。

2.2. 以太网的另类历史

以太网 Ethernet 这项技术由美国 Xerox 公司 PARC 部门的 Robert Metcalfe 等人受 ALOHA 的启发于 1972 年提出，术语“以太网”作为标准是在 1982 年由 DCE、Intel 和 Xerox 三家企业联合发布的，此时的 Robert Metcalfe 已经另起炉灶成立了 3Com，这是外话。早在 1993 年，以太网就已经统治了 TCP/IP 协议族中局域网技术。它使用的接入方法称为 *CSMA/CD*，全称为 *Carrier Sense, Multiple Access with*

Collision Detection，翻译成中文就是“载波监听，多点接入，冲突检测”，以太网从最初提供 2.94Mbits/sec 的速率（1M=10^6，而不是 2^{20}），当前大规模使用的以太网提供 10Gbits/sec 的速率，业界正在研究 40G 和 100G 的以太网，可以说以太网速率的演进是 2.94M→10M→100M→1G→10G→40G→100G，将来会发生什么，没有人能够知道。由于以太网是一种多点接入的链路层技术，所以必须要为每一个节点或终端分配一个标识（*identifier*），称为以太网地址（*address*），以太网地址长 48bit，通常称为 *MAC*（*Medium Access Control* 介质接入控制）地址，这个地址现在已经被人广为熟知，也被称为硬件地址（*hardware address*），其实不止是以太网，和以太网关系甚密的 WLAN 也使用相同格式的地址，实际上 WLAN 和以太网是共享同一个 48bit 地址空间。

以太网除了这些数字上的变化，难道就没有其它的变化吗？以太网为了实现更高速的传输，实际网络连接类型也发生了翻天覆地的变化。

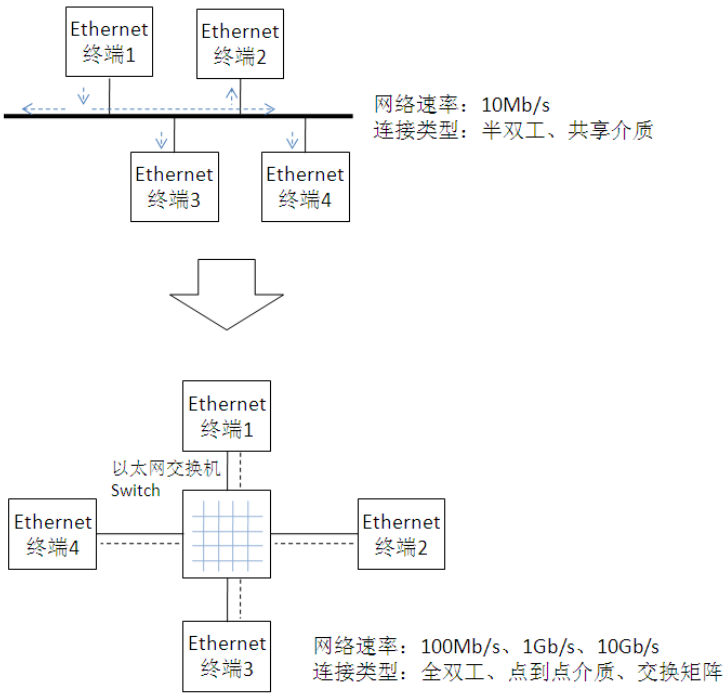


图 2-1 以太网从半双工共享总线模型发展到全双工、星型连接

在历史的车轮驶进 2000 年以前，以太网的特点如下：

1. 铜介质，原始的以太网使用同轴线缆，同轴线缆建设复杂，成本高昂，后人使用 8 芯双绞线（*4 Twisted pairs*）+*RJ45*（*Registered Jack 45*）连接器，配合 HUB 集线器组网（2 个 HUB 还可以通过双绞线互联，延伸介质范围），简化了以太网的建设；
2. 共享介质，链路就是介质，4 个终端连接在同一链路就意味着它们连接着相同的介质，那么如果所有终端同时发数据的话，就会在相同的介质上发生冲突，而以太网的 CSMA/CD 就是为了解决共享介

质上多点接入的冲突问题，使用 CSMA/CD 后可以保证多个终端在共享介质上发生冲突的概率得到控制；

3. 单介质，接口的收发使用相同的物理介质，意味着发送时无法接收，接收时无法发送，这种收发特点被称为半双工（*Half Duplex*）。

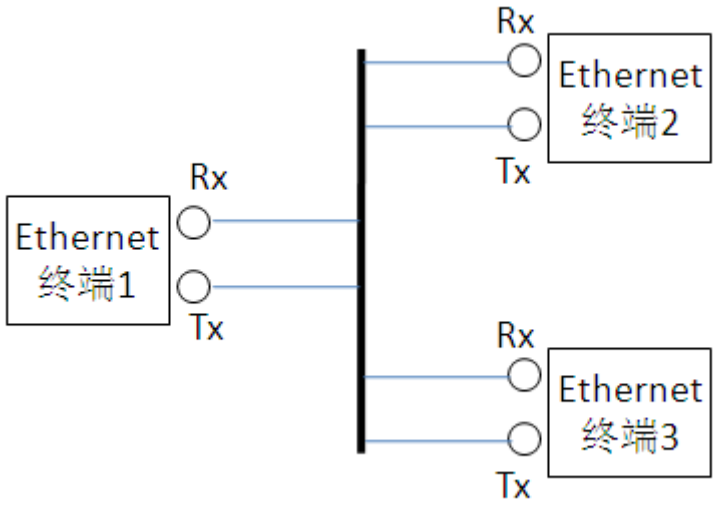


图 2-2 共享介质决定了半双工

图 2-2 说共享介质决定了半双工，怎么理解呢？在通信的物理层面上，只有终端 1 的 Tx 和终端 2 的 Rx 在同 1 个介质上时，终端 2 才能接收到终端 1 发送的数据，以太网是共享总线意味着终端 1 发送时，终端 2、3 都能够接收，从下面这张表可以得出以太网的终端收发只能共享介质，也就是单介质。

	终端 **1** 发送	终端 **2** 发送	终端 **3** 发送
终端 **1** 接收	未知	同一介质	同一介质
终端 **2** 接收	同一介质	未知	同一介质
中断 **3** 接收	同一介质	同一介质	未知

1. 从表中可以得出终端 **1** 发送和终端 **2**、**3** 接收同一介质；
2. 终端 **2** 发送和终端 **3** 接收同一介质；
3. 也就是说终端 **2** 接收及发送都和终端 3 接收相同介质，即终端 2 收发共享介质，只能是半双工；
4. 同理可以证明终端 1、3 也和终端 2 一样半双工，即表中的“未知”其实都是“同一介质”。

以太网发展至今，连接模式已经得到了非常给力的改善，从集线器演变而来的以太网交换机（*Switch*）在这种改善中起到了至关重要的地位：

1. 链路与介质分离，一条链路可以包含若干个介质，交换机提供若干端口，每端口提供 1 对介质（1 个收介质，1 个发介质），共享介质以太网可以说**“链路”**和**“介质”**是一码事；
2. 星型连接，星型连接按照端口划分后就是许多个点到点连接，即交换机每个端口连接 1 个以太网终端，所以各个以太网终端只是共享链路，但并没有共享介质，避免了共享介质的冲突问题；

图 2-3 点到点连接及收发双介质情况下的全双工

1. 全双工，刚才提到交换机的每个端口都提供 1 对介质，1 个负责接收数据，另 1 个负责发送数据，并且这个端口的介质因为点到点连接的关系，可以实现全双工（*Full Duplex*），及终端的收发任何时候都可以进行；
2. 交换矩阵，以太网还是个共享链路网络，及终端 1 要自由地和链路上的终端 2、3、4 通信，而不是只和其中的终端 2 通信，因此交换矩阵出现了，交换机的端口将所有的通信介质隔离，避免共享介质带来的冲突，而交换矩阵则将所有独立的介质联系起来，这个联系就像一个个开关（开关英文名为 Switch），当终端 1 要与终端 2 通信时，那么就将终端 1 的 Tx 与终端 2 的 Rx、终端 1 的 Rx 与终端 2 的 Tx 的介质开关打开，这个很好理解，那么终端 1 怎么在于终端 2 通信的同时要与终端 3 通信呢？这个同时是指人感觉上的同时，其实数据帧在 Tx 介质上是顺序发送的，也就是说交换机上的开关是根据每个数据帧做一次开关动作的，保证每个终端的 Tx 只和另外一个终端的 Rx 共享介质。

其实大家可能还会有问题，如果终端 2 和终端 3 同时向终端 1 发送数据帧，怎么办？首先交换机的交换矩阵处理速度要高于以太网速率，其次交换机还有适当的缓存，以存储没有及时发送的数据帧。可以说以太网交换机的出现，推动了以太网高速向前推进，而以太网的发展又推动了互联网的发展。

以太网交换机是网络术语“网桥”（*Bridge*）的一种，如今“网桥”的准确定义应该是将不同的通信介质连接到一块，忽略介质类型，使不同介质上的链路层协议互相通信，为网络层提供同一逻辑通信链路。这些通信介质可以是同质，也可以不同质（如分别使用双绞线和光线），可以运行相同的链路层协议，也可以不同（如以太网和 WLAN）。以前的“网桥”定义是连接不同的物理链路，而物理链路其实指的就是介质，本质上并没有大的修改，只是本文为了消除表达上的混淆，将物理链路使用“介质”来表示，而链路则指的是网络层协议所运行的逻辑链路，在介绍到网络层子网概念时，我们会了解到，一个子网所覆盖的网络通常就代表着一条逻辑链路。

2.3. 以太网与 IEEE 802 封装

上一节说到 1982 年三巨头 DCE、Intel、Xerox 联合发布了 *Ethernet*（翻译成“以太网”，“以太”曾被认为是光在宇宙中传播所使用的介质，尽管从未发现过这种物质）标准，过了一些年，美国标准化巨头单位 IEEE（Institute of Electrical and Electronics Engineers 电子电气工程师协会）802 委员会（该委员会的最初使命是引领全球局域网链路技术发展）发表了一系列稍微有点不相同的标准。802.3 覆盖了 CSMA/CD

网络（原生的以太网也使用 CSMA/CD），802.4 则覆盖令牌总线，802.5 覆盖令牌环，他们三都遵从 802.2 标准所定义的 *LLC*（*Logical Link Control*）协议，目前所有的 802 网络（例如 802.11 是 WLAN、802.15 是蓝牙、802.16 是 WiMAX）都遵从 802.2，还有一个 802.1 则是指定 802 网络中的管理，如 802.1X、802.1D、802.1Q（802 体系中的大写字母表示不依赖于其它 802 规范，小写字母表示依赖于某种特殊规范，如 802.11n 依赖于 802.11，802.1p 依赖于 802.1Q）等。通常链路层可以按照如下层次理解:

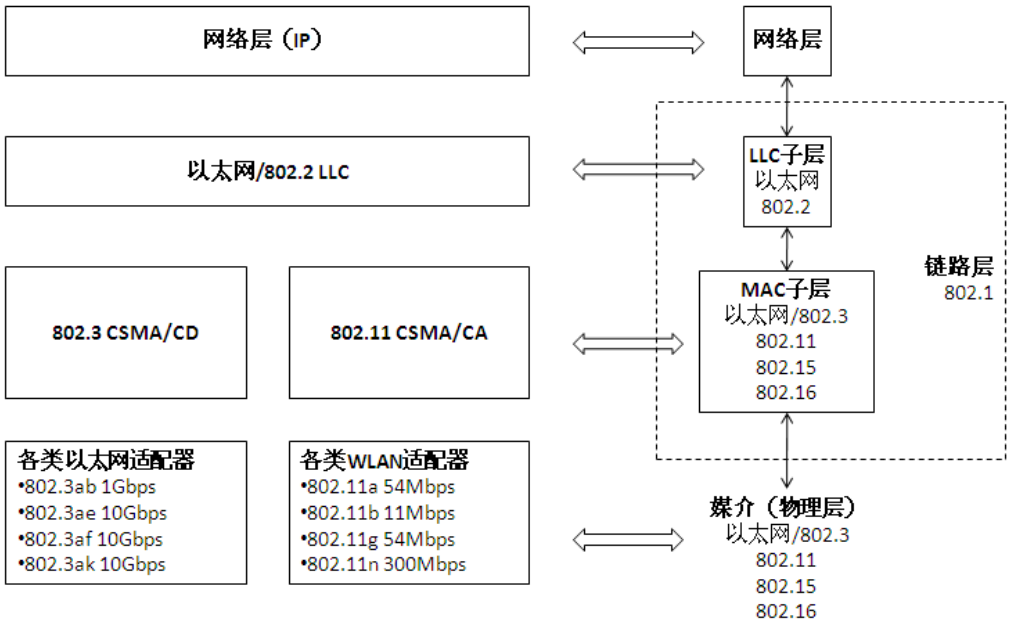


图 2-4 IEEE 802 委员会制定的 LLC 子层和 MAC 子层及相关标准

802.2 定义的 LLC 子层对网络层提供抽象的链路层驱动，屏蔽底层网络硬件类型的区别；而涉及与网络硬件类型、介质信号收发的工作交给 MAC 子层和物理层负责，802.3、802.11、802.16、802.17 都是 MAC 子层协议，同时 802 委员会又专门为各不同的 MAC 层制定物理层工作方式，如 802.11 的物理层就包括 802.11a/b/g/n。这种层次模型中不管物理网络类型如何变化，TCP/IP 的网络层不需要作过多修改的原因。IEEE 定义的 802.3 和 3 巨头定义的 Ethernet 在媒介类型、网络硬件类型都是相同的，也就是说 MAC 层是一致的，差别在于 LLC 子层，也就是数据包封装格式，目前 IEEE 的 802.3 制定 CSMA/CD 标准其实是兼容标准，两者在实际应用上差别甚小，由于目前市场的主流还是原生的 Ethernet，以至于说到以太网的时候既包括原生的 Ethernet 也包括 802.3（为了描述方便，本文中英文 Ethernet 指的是 RFC 894 中定义的原生 Ethernet 封装格式，中文“以太网”泛指 802.3 和 Ethernet），估计和 802.3 念起来比较费劲也有关系。

RFC 894 定义了 IP 数据包在 Ethernet 中的封装，RFC 1042 则定义了 IEEE 802 网络封装 IP 数据包格式。

主机需求 RFC 对连接以太网线缆的主机做了如下要求：

1. 必须能够接收和发送 RFC 894 封装的数据帧；
2. 应该能够接收 RFC 1042 封装和 RFC 894 封装相混合的数据帧；
3. 也许能够发送 RFC 1042 封装的数据帧，如果 2 种封装格式都能发送，那么必须有配置开关选择两种格式，但默认必须是 RFC 894 格式。

此标准一定，RFC 894 封装立刻压倒 RFC 1042，成为目前最常用的格式，这也可以说明为何我们通常称 Ethernet，而不称 802.3。那么两者在 LLC 层的差别到底在什么地方呢?

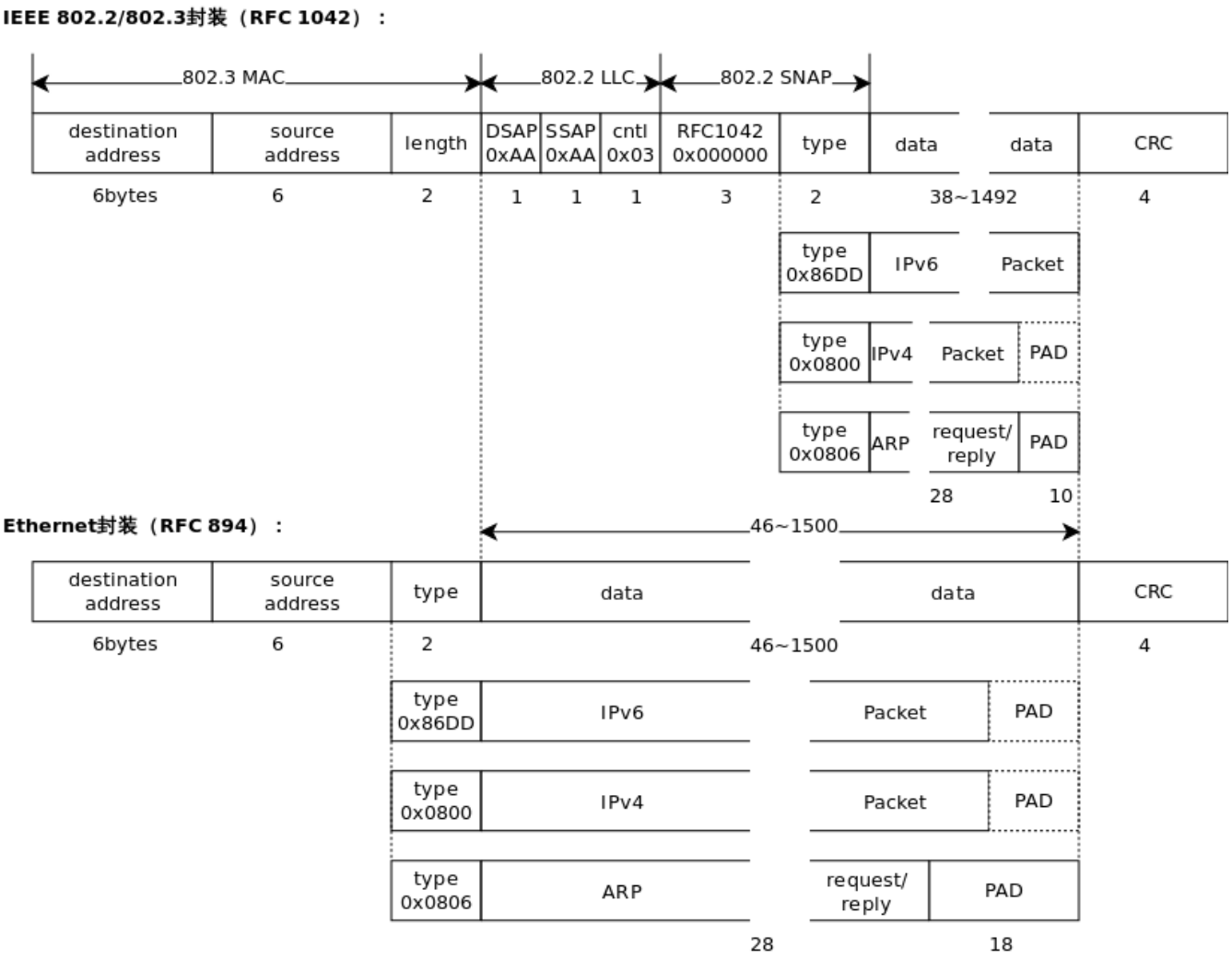


图 2-5 IEEE 802.2/802.3 封装（RFC1042）与 Ethernet 封装（RFC 894）的区别

- 两种封装格式使用相同格式 *destination address*（目的地址）和 *source address*（源地址），也就是 MAC 地址。而 ARP 用于 32bit 长 IPv4 地址映射到 MAC 地址，IPv6 ND（*neighbor discovery* 邻居发现）用于 128bit 长 IPv6 地址映射到 MAC 地址；
- 后面的 2 个字节在两种封装中有所不同，802 是 *length*（长度）字段，单位字节，其长度范围包括 *DSAP~data*（数据）部分，不包括 *CRC* 校验；Ethernet 则是 *type*（类型）字段，用于指明应用数

据类型，如 0x0800 表示 IPv4，0x0806 表示 ARP，0x86dd 则表示 IPv6；802 封装的类型字段则在则出现在 *802.2 SNAP*（*Sub-Network Access Protocol*）中，需要说明的是 802 封装的 *length* 和 Ethernet 封装的 *type* 取值范围不相同，因此很容易区分两种封装格式。

- 在 Ethernet 封装中，*data* 紧跟着 *type*，而 802 封装则是 3 字节长的 802.2 LLC 和 5 字节长的 *802.2 SNAP*（*Sub-Network Access Protocol* 子网访问协议）；*DSAP*（*Destination Service Access Point* 目的服务接入点）和 *SSAP*（*Source Service Access Point* 源服务接入点）都被设置为 0xaa，*CNTL*（*Control*）字段被设置成 0x03，3 字节长的 *org code* 字段被设置成 0x00-00-00；接下来就是 802 封装的 *type* 字段，其定义和 Ethernet 封装的 *type*（在 RFC 1700/RFC 3232 中定义了许多 *type* 值及含义，有兴趣的话可以一看）一致。
- *CRC*（*Cyclic Redundancy Check* 循环冗余校验）字段是一个校验值，它也有可能被称为 *FCS*（*Frame Check Sequence* 帧校验序列），用于检测整个封装中其余字段是否在传输过程中出错。
- 802 封装的应用数据最小为 38 字节，最大为 1492 字节；而 Ethernet 封装最小为 46 字节，最大为 1500 字节，有时应用数据如 ARP（固定 28 字节长）达不到最小长度，因此会被添加 *PAD*（*padding* 填充）字段，PAD 字段是变长的，其目的是凑齐最小长度（如 ARP 在 RFC 1042 封装中 PAD 长 10 字节，在 Ethernet 封装中长 18 字节）或者凑齐 8 的整倍数长度，PAD 字段会被 LLC 层完整地交给网络层或者 ARP 等链路层协议，在网络层协议或 ARP 进行处理时会将填充字段去除。

2.4. 以太网及 802 网络的重要扩展

3 巨头对以太网最大的贡献是 Ethernet 封装格式（RFC 894），而后续的发展逐渐被 IEEE 802 委员会接管了，IEEE 802 委员会主导了以太网的发展，比较重要的扩展有：

- 802.1D：MAC bridge 即定义 802 网桥；
- 802.1s：生成树（*Spanning Tree Protocol*），构造一个无环以太网；
- 802.1Q：*VLAN*（*Virtual LAN* 虚拟局域网）和 802.1p 优先级（*Priority*）；
- 802.1X：以太网接入的安全认证，原生的以太网是一个自由接入网络，没有任何安全措施，引进 802.1X 后，接入方式变成身份认证和授权接入的安全模式，局域网的安全已经变得越来越重要；
- 802.1AB：*LLDP*（*Link Layer Discovery Protocol*），链路层发现协议，用于同一链路上以太网邻居互相交互系统及端口配置信息；
- 802.1ad：*Q in Q*（801.1Q in 802.1Q），是一种较老的以太网嵌套技术，主要面向运营商领域，已经快不能满足 Carrier Ethernet（运营商以太网）要求，将要被下面的 802.1ah 取代；
- 802.1ah：*PBB*（*Provider Backbone Bridge*），是目前比较受关注的运营商以太网解决方案，它还衍生出 *PBT*（*Provider Backbone Transport*）解决方案；

- 802.1aq：*SPB*（*Shortest Path Bridge*），是目前比较受关注的数据中心以太网解决方案，它改变了传统网桥学习 MAC 地址和转发的规则，可以实现二层转发 *ECMP*（*Equal Cost Multi Path* 等价多路径），SPB 出现后，STP 类技术将慢慢被淘汰出数据中心；
- 802.3ab：双绞线铜介质上的 1Gb/s 以太网；
- 802.3ad 和 802.3ax：在并行链路上的链路聚合，具体协议是 LACP（*Link Aggregation Control Protocol*），如可以将 3 个 1Gb/s 的端口捆绑起来，变成一个虚拟的 3Gb/s 端口；
- 802.3ae 和 802.3af：光纤介质上的 10Gb/s 以太网；
- 802.3af 和 802.3at：*Power over Ethernet*，简称 PoE，在以太网双绞线上提供电源（802.3af 每端口提供最大 12.95W，802.3at 提供 25.5W），PoE 在无线 AP 部署、IP 话机部署上使用广泛；
- 802.3ah：*PBT*（*Provider Backbone Transport*），也是目前比较受关注的运营商以太网解决方案之一；
- 802.3ak：在双轴线铜介质上的 10Gb/s 以太网；
- 802.3av：10Gb/s EPON；
- 802.3az：节能以太网，即端口能耗可以根据端口带宽动态调节，以适应全球的节能趋势；
- 802.3ba：40Gb/s 和 100Gb/s 以太网。

这些重要扩展大都在广泛地使用在世界各地的以太网中，但对于 TCP/IP 协议族而言，都不是 *LLC* 子层的内容，而是属于 MAC 子层或同层范围，并不影响网络层的运行，在本文中将对几个重要的扩展如 802.1Q、802.1X 进行介绍。

2.5. WLAN（无线局域网）

前面大略提了一下 *WLAN* 差不多是以太网的无线电介质版本（我刚刚参加工作的时候甚至谑称 WLAN 为“空中 HUB”），以太网脱胎于 *ALOHA*（1960 年代由美国夏威夷大学主导研究的一种共享无线网络技术，ALOHA 是夏威夷原住民语 Hello 的意思），*ALOHA* 就是一种无线技术，所以 *WLAN* 和 ALOHA 也有很深的渊源。WLAN 标准化工作开始于 1997 年，而 Richard 写书是在 1993 年，所以当时他并没有研究这项技术。WLAN 的标准化技术是由 IEEE 802.11 委员会领导的，通常我们认为 WLAN = 802.11 = WiFi，但最耳熟能详的是时髦的 WiFi，稍微专业一点的知道 WLAN，更专业的才知道 802.11。像介绍以太网连接模式那样，这里介绍 WLAN 连接模式。

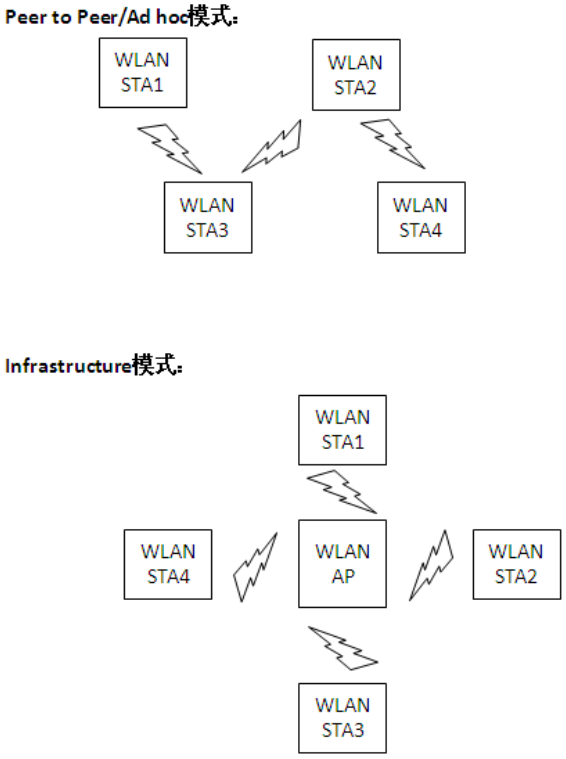


图 2-6 WLAN 的两种连接模式

WLAN 的通信介质是同频无线电（通常被划分为不同频段的 *Channel* 信道），无线电是在自由空间传播的，无线电通信具备开放接入的特点，也是一种共享介质模型，因此也存在冲突的可能，WLAN 解决多点通信冲突的解决方法是 CSMA/CA（载波监听、多点接入、冲突避免，A Avoidance 避免），为何不是 CSMA/CD 呢，因为 WLAN 的无线信号在开放空间内传输，信号并不强（大多 WLAN 设备在开放空间全向覆盖半径为 100m，特殊情况使用定向天线可达 10km），能量容易被金属等材料吸收，有可能无法覆盖到空间，载波介质在空间中可能是断续的，而不是连续的，因而无法保证所有节点都可以互相监听，而以太网则在专门的线缆内传递，线缆上的节点都能够进行监听，如图 2-6 中 *Infrastructure* 模式中终端 4 和终端 2 因为距离较远可能就无法互相监听，也就检测不到冲突，因此无线电介质上多点接入的冲突主要是靠事先避免（Avoidance），而非事后检测（Detection）。在 WLAN 中，所有节点被称为 *station* 简称 STA，在 *ad hoc*（此为拉丁语，意为特殊）模式中，所有 STA 都是对等的，所以也叫 Peer to Peer 模式；在 Infrastructure 模式中，有一个特殊的 STA 称为 *Access Point*（无线接入点，简称 AP），其余 *STA* 都要与 AP 建立连接，STA 之间的通信都要由 AP 转发。两种模式中，Infrastructure 模式更加常见，WLAN 在这种模式中也扩展出了极其丰富的应用。总体来说这两种模式的差别在于转发上：

- ad hoc 模式中，以图 2-6 为例，因为信号强弱的缘故，STA1 无法检测到 STA4，只能检测到 STA2，那么 STA1 只能与 STA2 通信，而无法与 STA3、4 通信，也就是说 ad hoc 模式中，通信无法中转；

- Infrastructure 模式中，所有的 STA 必须连接到 AP，如图 2-6 中，STA1、2、3、4 皆连接到同一个 AP，那么 STA1 要和 STA4 通信时，即使 STA1、STA4 因为信号足够好，STA1 也要讲数据帧交给 AP 来转发，也就是说 Infrastructure 模式中，所有数据都必须由 AP 进行中转，AP 就是所谓的 Infrastructure（基础设施）中的一部分，在 WLAN 中，完整基础设施的名称是 *distributed system* 分布式系统，现代的大型无线网络中，一个 DS 通常由若干 AP 以及连接这些 AP 的有线网络、AP 控制器构成（AP 控制器可以控制若干个 AP 共同覆盖一个比较大的区域范围）。

IEEE 802.11 委员会针对 WLAN 所做的标准非常多，有涉及介质层面的 802.11a/b/g/n，也有 MAC 层面的 802.11i/e 等等，除此之外，无线信号相关的射频理论知识也有一个非常庞大的体系，可以说比 TCP/IP 框架复杂专业得多。限于能力和时间原因，不在本文中专门介绍，读者可以从已出版的书籍中找到相关知识。

2.6. WLAN 封装

WLAN 中和 IP 打交道的是依然是 LLC 层，介绍 LLC 层，从图 2-4 来看，802.11 和 802.3 共用相同的 LLC 封装，差别在于 MAC 层（使用 tcpdump 以及 winpcap 工具捕获的数据包其实都被解析成 Ethernet 的 MAC 封装格式，看不出各个链路层协议封装的差别，所以如果要真正查看 802.11 的 MAC 封装的数据帧，需要使用专门的无线电空中接口抓包软硬件套装）。下面就介绍 WLAN 的 MAC 封装格式，如图 2-7 所示。

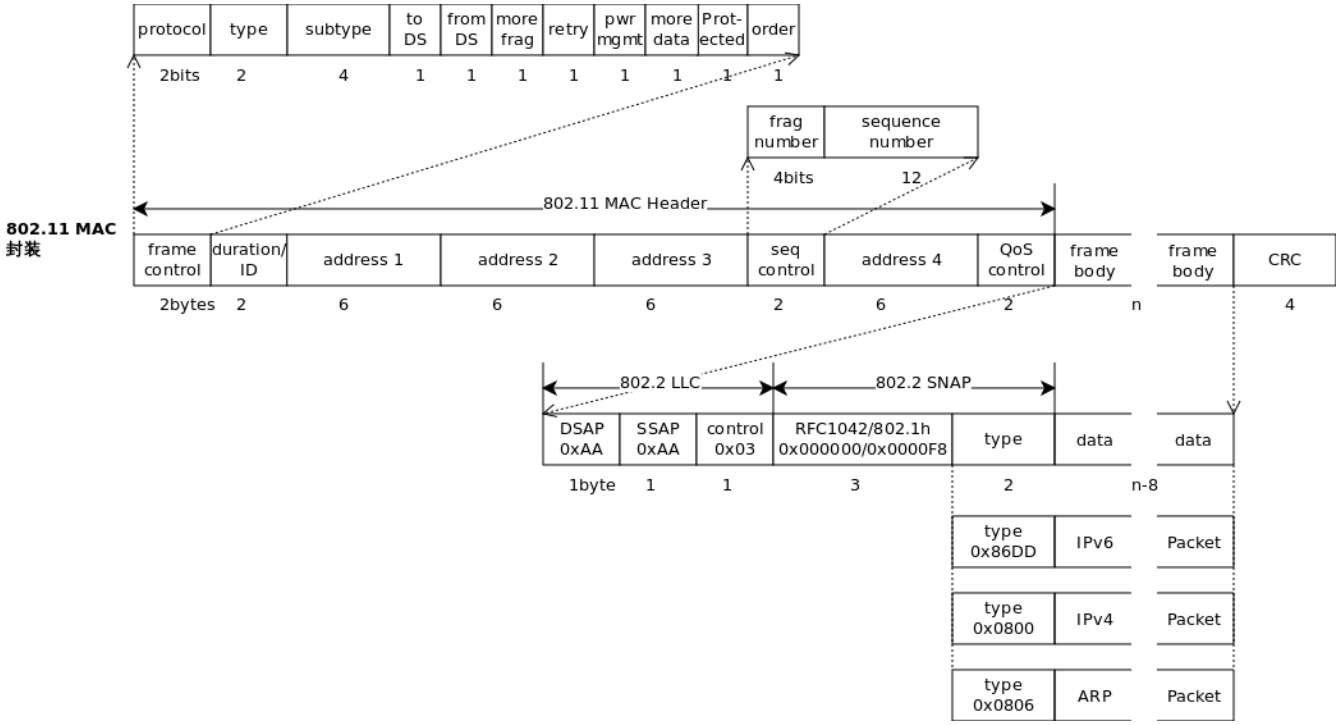


图 2-7 802.11 定义的 WLAN MAC 层封装示意

从上图来看，802.11 的 MAC 层封装与以太网相比复杂了许多，多出来的这么多字段原因在于以太网和 WLAN 的传输介质物理特性差异所致，WLAN 需要更多的控制以实现高效地传送，降低冲突。由于 WLAN 是完完整整由 IEEE 802.11 委员会主导制定的，故其 LLC 层封装采用了标准的 802.2 LLC 封装格

式，和图 2-6 中 802.3 封装是一致的。下面对各个字段进行解释：

1. *Frame Control* 帧控制，该字段 2 字节长，分为 11 个子字段：
   1) *Version* 版本，2 位长，目前值为 0，其余值供未来使用；
   2) *Type* 类型，2 位长，目前有 0、1、2 三个值，分别对应 *management frame* 管理帧、*control frame* 控制帧、*data frame* 数据帧，管理帧用于加入、退出 WLAN 时的关联；控制帧，通常和数据帧配合，用于无线信道（无线电介质在链路层的马甲）清空（*area clear*）、占用（*channel acquisition*）和监听（*carrier-sensing maintenance*）、数据帧确认（*positive acknowledgment of received data*）；数据帧是个苦力，负责搬运网络层交付的各种数据，也是我们使用 WLAN 的目的所在；
   3) *SubType* 子类型，4 位长，用于协助类型字段，不同的子类型可以用于不同用途；
   4) *ToDS*：从 STA 到 DS，置 1 表明数据帧是从 STA 经由 AP 发送到 DS；
   5) *FromDS*，从 DS 到 STA，置 1 表明数据帧是从 DS 经由 AP 发给 STA 的；如果 ToDS 和 FromDS 字段同时置 1，表明这个数据是在 DS 系统内部传输（无线桥接），而非 AP 和 STA 之间或 STA 之间传输；若同时置 0，则表示此时 Station 工作在 ad hoc 模式中；
   6) *More Fragment* 更多分片，置 1 用于通知对端该数据帧只是数据帧的众多分片之一，后面还有分片没发完，需要对端做好准备，WLAN 为了提高线路利用率避免重传，会将较大的数据帧分片发送，即使发生重传也只是重传分片；
   7) *Retry* 重传，置 1 表示该数据帧是重传的，WLAN 是一种可靠链路层协议，大多数帧需要肯定确认，若没有收到确认，需要重传，这和 1993 年时的以太网是很格格不入的一个理念，主要原因是无线共享介质容易受到干扰，可靠性设计更能提高介质利用率，实际上 802.3 委员会也针对数据中心局域网发布了可靠以太网规范，不需要对此感到太奇怪；
   8) *Power management* 能量控制，WLAN 在设计之初就考虑到了这个东西会用在笔记本上，甚至用在手机上，因此无线信号节能是一个非常有用的技术，当 STA 的上层协议认为没什么数据要发送时，会把最后一个帧的“能量控制”位置 1，然后进入节能（*Power Save* 简称 *PS*）状态，暂时关闭 WLAN 收发，但 AP 是无线接入基础设施，此位不得置 1；
   9) *More data* 更多数据，配合 *power management* 位使用，“能量控制”是给 STA 使用的，那么“更多数据”就是给 AP 使用的，AP 将此位置 1，用于提醒 STA，AP 还有不少数据还在缓存中没发完，不要急着进入节能状态；
   10) *Protected* 帧保护，可能大家听说破解 WLAN 密码时会知道有个 WEP（Wired Equivalent Privacy 有线对等加密）破解，这个密码就和这个 Protected 位有关，当该位置 1 时表明数据帧已经被加密，为 0 则表示数据帧是明文（没有被加密的数据在密码学 *cryptography* 中被称为明文 *Clear text*，反之称为密文 *Encrypted text*）传送，因为 WLAN 安全问题，大多 WLAN 都是密文传送；

11) *Order* 保序，置 1 时指所有接收的数据帧都要严格按照先后按顺序处理；

2. *Duration/ID* 回合/标识，2 字节长，这是一个比较奇怪的字段，有 3 种用法：
   1) 最后 1 位为 0，此时意思为回合，即帧的发送方预期此帧发送完整过程需要占据介质多长时间，单位是 us（微秒），通知同一个无线介质上所有监听节点（不管是 STA 还是 AP）发送端需要占用介质的时间长度，监听节点通过监听所有 802.11 帧中的 Duration 以计算通信介质当前是忙碌还是空闲，以避免发生介质访问冲突；
   2) 最后 1 位为 1，其余位为 0，说明该帧是在 *CFP*（*contention free period* 无竞争周期）发出的，用于通告其余节点，避免干扰无竞争帧的传送；
   3) 最后 2 位为 11，在 STA 从 PS 模式苏醒后，会发送一个 *PS-Poll*（节能-轮询）帧，以便从 AP 取得尚在缓存中为发送的帧，此时该字段为 *association ID*（关联 ID，简称 *aid*；STA 连接到 AP 中有一个过程叫关联，并从 AP 获得一个 aid），用于提醒 AP，在缓存中找到该 STA 相关联的数据帧。
3. 地址 1、2、3、4 字段(oh my ladygaga)，WLAN 的地址字段竟然有 4 个（以太网只有 2 个地址字段），WLAN 的地址字段格式和以太网一致，这些地址段填写可以是如下选项，每个字段根据帧控制字段 Type、SubType、ToDS 和 FromDS 字段的不同，有不同的含义，下表是针对数据帧及 ToDS、FromDS 字段值所对应地址字段含义：

功能	**ToDS**	**FromDS**	**Address1 (Receiver)**	**Address2 (Transmitter)**	**Address3**	**Address4**
IBSS	0	0	DA	SA	BSSID	未使用
To AP	1	0	BSSID	SA	DA	未使用
From AP	0	1	DA	BSSID	SA	未使用
WDS	1	1	RA	TA	DA	SA

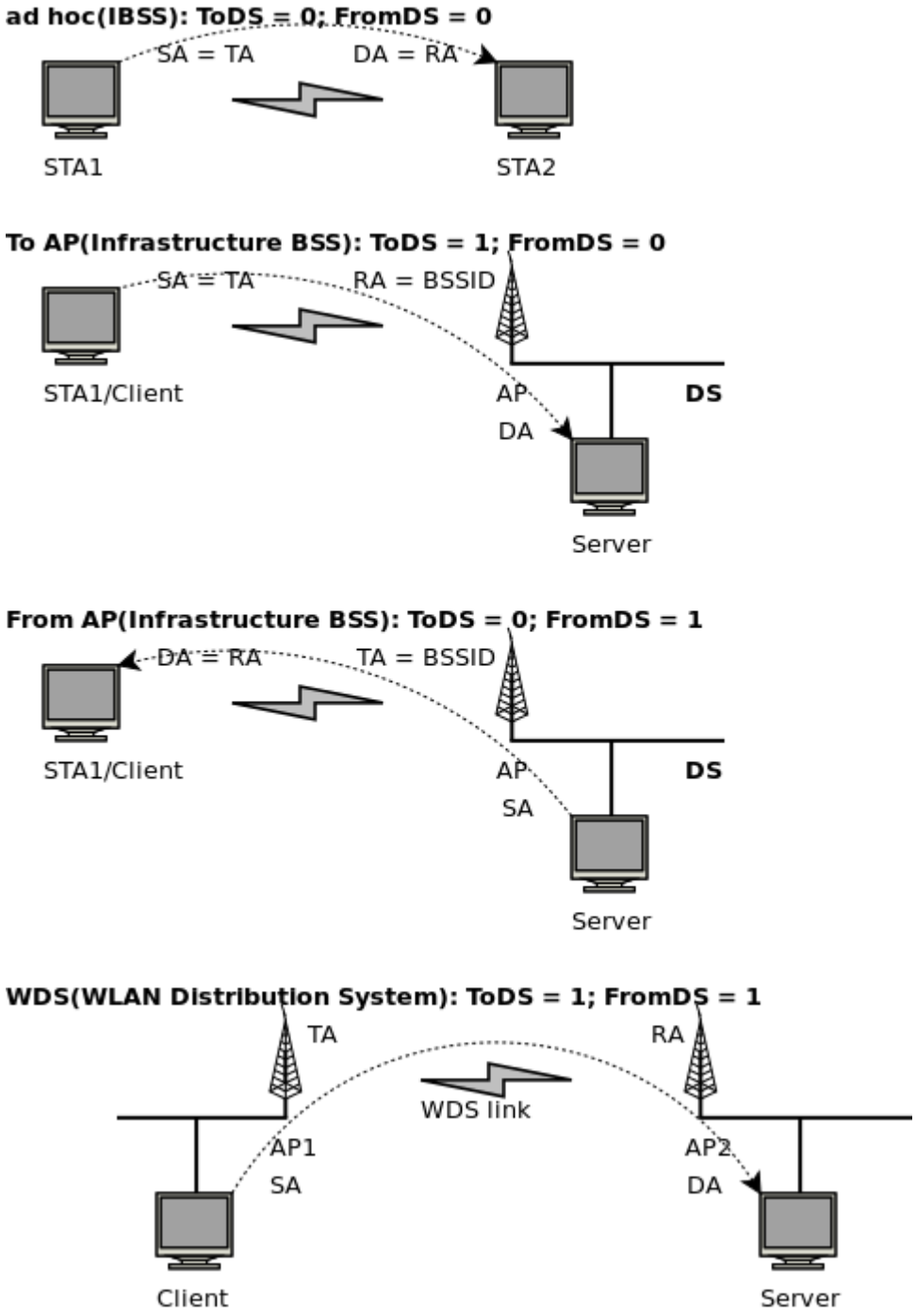


图 2-8 WLAN 不同情况数据帧地址字段含义

结合 2.8 和上表，WLAN 封装在 MAC 层的地址变化多端，相当复杂，每种地址含义如下：

1) BSS ID，BSS 称为（Basic Service Set），指在一个相同的无线通信介质，各 STA 要互相通信共同遵守的能力集合，在 ad hoc 模式中，BSS ID 通常是随机产生的；在 Infrastructure 模式中，AP 作为该无线信道 BSS 的提供者，该 AP 的 MAC 作为其辖区范围内各 STA 的 BSS ID，在 Infrastructure 模式中 STA 之间的数据帧要由 AP 中转，所以 AP 的 MAC 经常以 BSS ID 的形式出现在帧头；

2) Destination Address（DA 目的地址），最终需要接收该数据帧的 STA 或位于 DS 中节点的 MAC 地址，数据帧只有到达 **Destination** 才会交给上层协议（如网络层协议 IPv4、IPv6）处理；
3) Source Address（SA 源地址），最初发出该数据帧的 STA 或位于 DS 中节点的 MAC 地址；
4) Receiver Address（RA 接收者地址），在同一个无线介质中直接接收该数据帧的 MAC 地址，RA 未必是 DA，也就未必要交给上层协议处理，在图 2-8 中 To AP 部分，RA 是 AP 的 MAC 地址（也是 BSSID），但 AP 收到帧后，根据 DA 將数据帧转发给 DA，AP 的转发工作只是在链路层面，从这看 AP 是一个网桥；
5) Transmitter Address（TA 发送者地址），在一个无线介质中发送该数据帧的 MAC 地址；
6) 大多数数据帧拥有 3 个地址字段，这也是为何地址 1、2、3 连续排列，但上表统计其实并不完整，有的控制帧和管理帧甚至只有 RA，具体可以查阅 802.11 相关规范。

4. *Sequence Control* 序列控制，是用于无线数据帧分片与重组使用的，由 2 个子字段组成，分别是 12bit 长的 *Sequence Number*，用于标识每一个完整的未分片数据帧，从 0 开始计数，每发一个帧，其 Sequence Number 增 1，直到 4095 后翻转至 0；4bit 长的 *Fragment Number*，用于标识分片后的数据帧在未分片数据帧中的位置，从 0 开始计数，即一个数据帧最多可以被分成 16 片；
5. *QoS*（*Quality of Service*）*Control* 控制，这是在 802.11e 中专门定义的字段，是个可选字段，由帧控制中类型/子类型联合定义，QoS 支持能力是由信道中通信双方协商后确定的，包含优先级、确认策略、载荷类型、访问类型等子字段，802.11e 是处理共享链路中的服务质量问题；
6. *Frame data* 帧数据，采用标准的 802.2 LLC 封装格式，和 802.3 不同的是对数据部分长度的要求，数据部分长度并没有明确规定，根据不同的 802.11 物理层有不同的取值范围，如在 802.11n 之前 data 取值是从 0～2312，到了 802.11n，最大取值范围竟然可以达到 7955 甚至 11414（是不是该叫一下 oh yeah, baby！），但网络层发送给 LLC 层一般都是在 1500 字节以内，802.11 的 frame body 之所以可以那么大，主要是为了 MAC 帧聚合以提高媒介利用率，802.11 很特殊的一点还有没有 PAD（填充）字段。图中 2.7 中对封装 IPv6 和 ARP 做了举例；
7. *CRC* 校验，这个和以太网的 CRC 是完全一致的。

2.7. WLAN 的重要扩展

互联网催生了人类有史以来最便捷的沟通方式，技术从实验室走到了街边地摊，网络也从笨重的电脑蔓延到了手机等超级移动设备，WLAN 这种便捷的接入方式，无疑很受笔记本、超级移动计算机、甚至手机用户的青睐，假设每个家庭都要一台带 WLAN 功能的路由器，全球的家庭用户估计要 10 亿台 WLAN 设备的需求，再加上企业、政府等等的需求，这块市场吸引了许多专家投入精力进行研究，WLAN 上的扩展可以说一点也不比 802.3/以太网逊色，但大部分都是和无线射频、频谱划分等物理层特性，下面是收集的一些和本

文内容比较相关的 2 个扩展：

- 802.11e：QoS 增强特性特性，用于解决 WLAN 中的数据突发和多点接入中的传输效率；
- 802.11i：安全增强特性，定义了 WLAN 的安全框架，在 802.1x 的基础上引入了 *WPI*。

2.8. PPP（Point-to-Point Protocol）点到点协议

有人经常把 PPP 和 P2P 混淆，其实了解字面意思后就知道二者之间风马牛不相及，PPP 是一种链路层协议，P2P 则只是一种通信模型。之前介绍的以太网、WLAN 都是多点接入链路技术，以太网尽管实现了点到点介质，但链路依然是多点接入的。

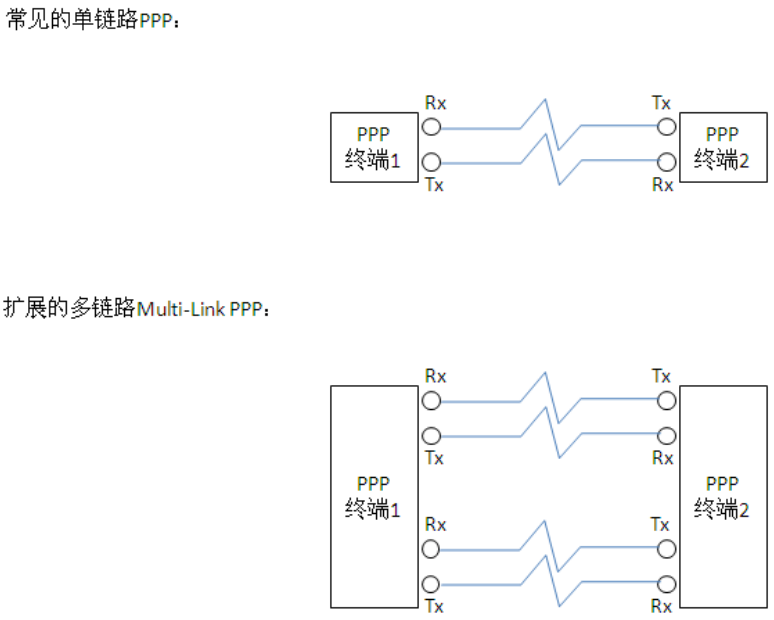


图 2-9 PPP 链路连接模型

而 PPP 就是专门为点到点链路设计的协议，介质是点到点的，链路也是点到点的。如图 2-9 所示，PPP 的物理介质有可能不是连续的，但 PPP 链路建立之前要求先建立连续通信介质。这可能听起来有点危言耸听，怎么建立连续介质呢？我们忽略了人在网络中的重要性，网络是人设计的，当然我们有建立连续介质的权利，比如电话，要求人拨电话号码，拨号就是一个建立连续介质的过程，可以说 PPP 链路和以太网很不一样的特点就是“按需接入”，而以太网、WLAN 都是“自由接入”特点，因此使用 PPP 协议的物理层通常是 Modem、广域网专线这样的场景。PPP 的链路层模型可由图 2-10 来描述。

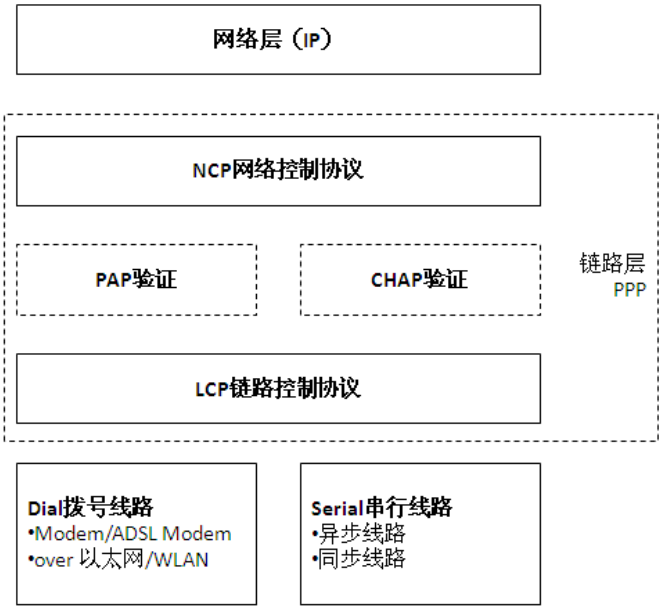


图 2-10 PPP 链路层次模型

PPP 链路层次模型如图 2-10 所示：

- 媒介和物理层已经介绍了拨号（*Dial-up*）线路，该线路最明显的就是传统 Modem 拨号和它的升级版本 ADSL Modem 和 3G 拨号，以及虚拟拨号，既在以太网/WLAN 上建立虚拟拨号链路，我们称为 *PPP over Ethernet*，简称 *PPPoE*；拨号线路的特点可以建立长距链路（如建立从南国广州到东北哈尔滨的链路），但是带宽小，而且不稳定，只能适合个人用户，如果是企业或者研究机构，则要求带宽高一些，稳定的链路，这就是广域网中的串行线路，分同步、异步两种，目前同步 SDH 线路应用广泛一些，有通常比较常见的带宽选项有 2M、155M、622M、2.5G，当然带宽越高，要掏的银子也少不了；
- PPP 工作在链路层，最底层的是链路控制协议，*Link Control Protocol*，作用是建立、配置、测试网络数据连接，也就是说在物理层在人的干预下建立连续介质后，介质两端的 LCP 就要开工了，协商双方的参数是不是相互兼容，如果各种必要参数 OK，那么 LCP 宣告 PPP 链路已经建立好了，把工作移交给上一层；
- LCP 的上一层是认证模块，这是一个由 LCP 协商出来的可选模块，共有 2 种认证方式：PAP（*Password Authentication Protocol* 密码认证协议）和 CHAP（*Challenge Authentication Protocol* 挑战认证协议），这两种认证都采用用户名、密码认证，可以确认拨号者身份，两种协议的差别在于安全性，PAP 在线路上传输用户名、密码的明文，CHAP 则传送 MD5 计算后的摘要，比较安全；认证一般用在个人使用拨号线路中，目的是确保受信拨号，认证完了也可以计费收钱，而

串行线路是专线，也就是介质是无需拨号提前建好，专门给某两个终端使用，银子也是提前就付了的，使用专线的也基本上是大客户，认证没有必要；

- 认证通过或者免认证后就是 NCP（*Network Control Protocol* 网络控制协议）进行最后的收尾了，NCP 是一个家族的统称，PPP 为不同的网络层协议专门制定 NCP，如网络层是 IPv4，那么就叫 IPCP，如果是 IPv6，那么就是 IPv6CP，如果是 OSI，那就叫 OSICP，但 OSI 协议自出生就被 IPv4 牢牢扼杀住，能一睹其芳容的机会很少，除 OSI 之外还有一些被 IPv4/IPv6 干废的网络层协议有 DECnet、Appletalk 等，它们都在 PPP 上有自己的 NCP；NCP 的作用是协商链路两侧终端的网络层地址信息、压缩之类的参数，NCP 竣工后，网络层协议就可以在 PPP 链路上欢畅运行了。

PPP 的主要扩展包括 *Multi-Link PPP*（多链路 PPP）、PPP 压缩（*Compression*）等技术。

2.9. PPP 封装

PPP 的封装格式比以太网、WLAN 比起来简单得多，如图 2-11 所示。

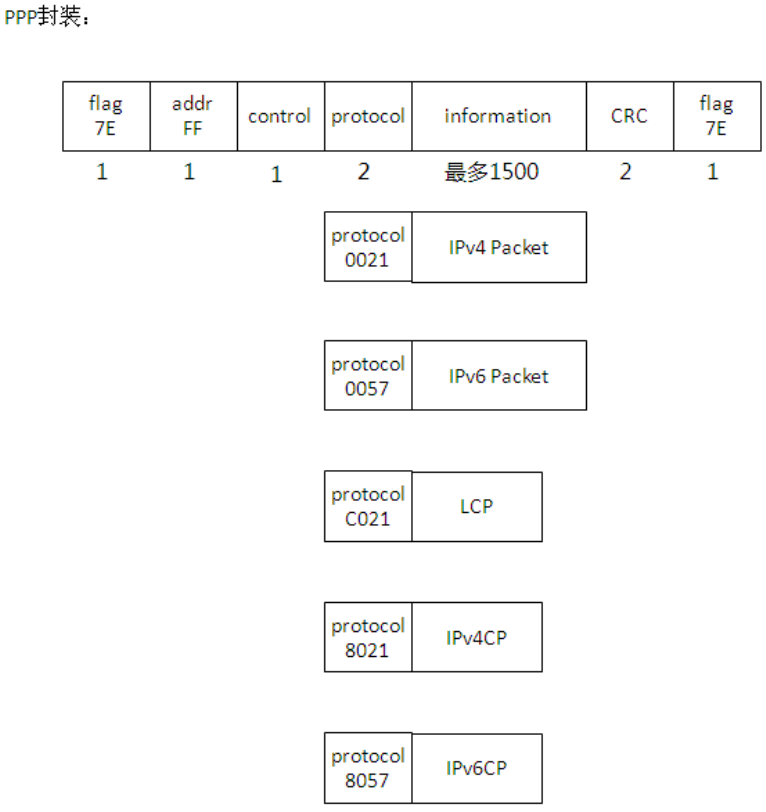


图 2-11 PPP 封装

- 所有 PPP 帧都是以 *flag* 开头，也以 flag 结尾，也就是说 flag 字段是 PPP 帧的间隔，这是一个固定值 7E；
- *address* 地址是固定值 FF，因为 PPP 链路是点到点链路（链路上就只有 2 个终端），并不是以太网那样的点到多点链路，如图 2-9，终端 1 发送只能是终端 2 接收，反之亦然，并不需要地址来区分这

2 个终端，也不需要地址来区分这个数据帧是发给哪个终端的；

- *control* 控制也是固定值 03；
- *protocol* 协议就不是固定值了，它的作用和以太网的 *type* 字段类似，用于指明 *information* 字段是何种协议数据，如 0x0021 是 IPv4 数据，0x8021 则是 IPv4CP，0x0057 是 IPv6 数据，0x8057 则是 IPv6CP，0xc021 则是 LCP 数据；
- *information* 信息字段类似于以太网的 data 字段，最长 1500 字节，最短可以为 0；
- *CRC* 字段和以太网的 *CRC* 字段作用一致，只不过只有 2 个字节，以太网的 CRC 是 4 字节。

PPP 帧的开头和结尾都是以 0x7E（二进制表示 01111110）为标志的，如果信息中的内容也存在 0x7E，此时就杯具了，PPP 的信息字段中 0x7E 被认为是数据帧的结尾，PPP 数据帧被截断，那么怎么避免这种误会呢？答案是 *escape character* 转义字符（直译的话叫“脱险字符”），即将信息中的 0x7E 用别的数值串来表示：

1. 单字节 0x7E 被转换成双字节 0x7D，0x5E，这是 *flag* 字段的转义；
2. 单字节 0x7D 被转换成双字节 0x7D，0x5D，这是转义字符的转义，有人可能会问如果信息中真的有 0x7D，0x5D 这个序列怎么办，根据程序的处理，它只处理 0x7D，并不管序列，因此会变成 0x7D，0x5D，0x5D；
3. 此外还对一些 ASCII 码中小于 0x20 的字符也做了转义定义，这些字符通常都是控制字符，在一些古老的设备中，这些特殊的控制字符会解释成特殊含义，目前的设备虽然可能没有这些问题，但为了兼容以往的设计，默认情况下，依然会对这些控制字符进行转义。

Stevens 在 1993 年写原版详解时，网络基本上是低速（当时的以太网还相当昂贵，主要使用 19.2kb/s 的串行线路）网络，对于封装效率非常的在乎，封装效率高说明链路带宽大部分用来传输数据，而不是封装字段开销。可以通过这个公式来计算“封装效率”＝“数据部分长度”÷“数据帧长度”，比如 PPP 中，数据部分就是 *information* 字段，最长 1500 字节，此时数据帧长度就是 1+1+1+2+1500+2=1507 字节，那么 PPP 封装效率接近于 99.536%，在当时还是可以接受的。

2.10. PPPoE（PPP over Ethernet）

我们知道以太网是一个开放式接入的网络技术，曾几何时，我们的电脑只要有网卡，有网线就能连接到校园网，然后开始免费冲浪，免费固然好，只是网速慢，以太网是个共享链路，100Mb/s 链路带宽让 200 人使用，人均不到 500kb/s 带宽，可以说人越多，网速越慢，这和北京、上海不限制发车牌是一个道理。那么网络解决方案也和北京、上海通过摇号发牌治堵如出一辙，不能再搞开放式接入了，要搞身份验证后接入，叫受信接入或者安全准入。

以太网开始设计的时候就没有受信接入这个概念，以太网是计算机通信的产物，能连上网络的基本上是研究机构、科学家，大家都是文明人，又那么熟，用不着受信连接，都连接起来，多热闹。随着以太网走向了民

间，大家发现这种自由网络让大家联机异常方便（以太网是可自建的局域网，PPP 则只能使用运营商的拨号线路），带宽还高（PPP 使用的拨号带宽通常是 2Mb/s 以内），此时运营商也闻着味进入了这个高带宽接入行业，运营商运营好网络的目的就是吸引用户，然后收钱，特别是按人头收钱，以太网这种开放接入铁定是吸引不了太多用户的，也就收不到钱，运营商也要求进行受信接入，因为运营商的套路很简单“受信是收费的前提”。

在这种背景下 PPPoE 诞生了，PPP 因为有 PAP、CHAP 认证功能，所以可以作为受信使用，而且在拨号链路上用得很成熟，直接把 PPP 架在以太网之上，一方面可以享受以太网高速接入带来，另外 PPP 可以提供受信接入。这也很好解释为什么没有人去搞 Ethernet over PPP。

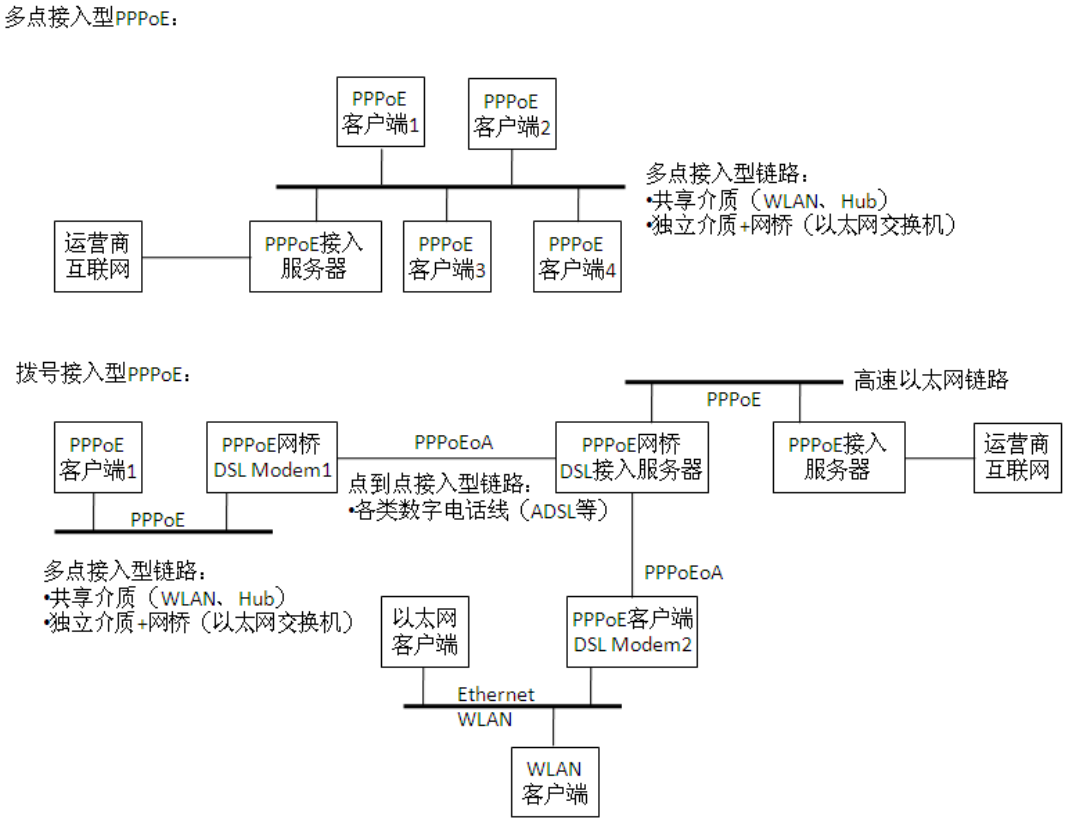


图 2-12 PPPoE 连接模型

常见的 PPPoE 有两种连接模型：

1. 多点接入型：若干 PPPoE 客户端和 PPPoE 接入服务器都接入到同一条以太网链路，要求每个局域网络都有一台 PPPoE 接入服务器；
2. 拨号接入型 PPPoE：PPPoE 客户端和 PPPoE 接入服务器通过点到点广域链路相连，而客户端通常是通过局域链路连接到 PPPoE 网桥（我们常见的 ADSL 宽带路由器等，准确的称呼是 *Digital Subscriber Line Modem*），因为以太网帧是无法直接在拨号数字线路上传输的，所以要把以太网帧封装到 DSL 线路上的数据帧（ATM *Asynchronous Transfer Mode* 封装，）中，也就是所说的 PPPoEoA，传送到远端的 DSL 服务器，这也是个 PPPoE 网桥，它将 PPPoEoA 的封装解成

PPPoE，通过高速以太网发送给 PPPoE 服务器，由 PPPoE 服务器统一为众多拨号客户端做认证。当前拨号接入型 PPPoE 要流行一些，运营商的大部分线路都是电话线路，把传统的电话线路改造成数字线路（DSL）的成本要比在一个小区打造以太网低（几乎所有的建筑都布了电话线，但我们无法保证所有建筑都布好了以太网），每条电话线安装一个 DSL Modem 的价格也不算离谱，用户可以接受 DSL 线路带宽所带来的花费。另外一个很给力的原因就是只需要统一的接入服务器即可，因为 PPP 是需要做身份确认的，统一接入显然有利于运营商统一计费。目前的状况是 DSL Modem 自带拨号功能，也就是说 DSL Modem 已经变成了一台路由器（所以我们经常说 ADSL 路由器），这么一来用户更加开心，PC 的网关是 DSL Modem，又是使用以太网、WLAN 这种多点链路相连，用户的笔记本、手机都可以连接到 DSL Modem，然后都可以连接到互联网，图 2-12 中拨号接入型 PPPoE 的 DSL Modem2 就是这种情况。

以太网接入型 PPPoE 的目的是解决以太网的开放接入问题，现在正逐渐被专门的局域网安全准入技术所替代，这就是我们后面要介绍的 802.1x 和 802.11i。在结束前，我们再来看一下 PPPoE 的封装。

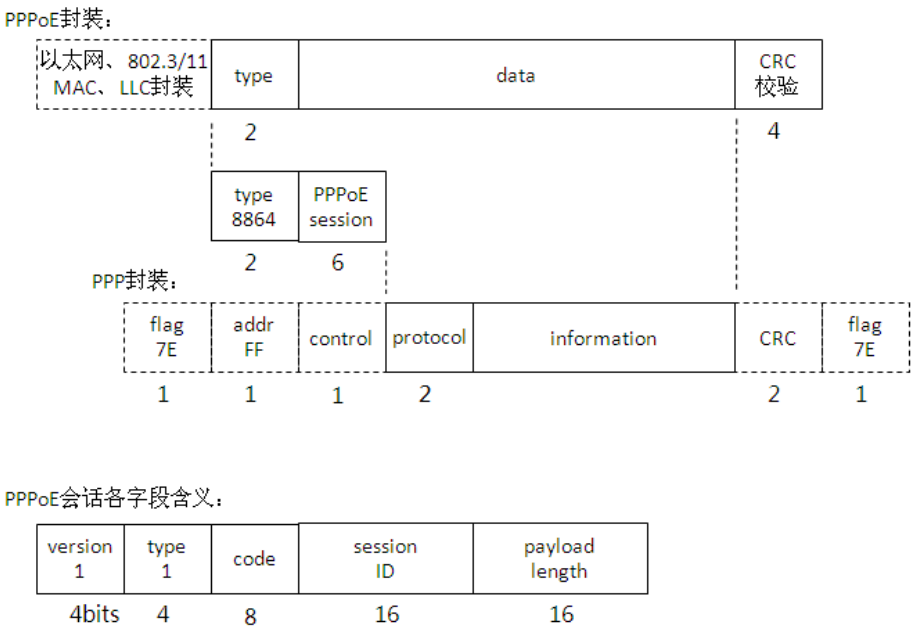


图 2-13 PPPoE 封装

PPPoE 可以运行在以太网、802.3、802.11 链路之上，故封装从这 3 种链路 LLC 头中的类型开始介绍：

- PPPoE 帧使用以太网 *Type* 值有 0x8863（PPPoE *discovery* 阶段）和 0x8864（PPPoE *session* 阶段），和 IPv4 是 0x0800，IPv6 是 0x86dd 一样；
- *PPPoE Session* 会话，是 PPPoE 封装中最核心的字段，它由下列子字段构成：

- *Version* 版本，4bit 长，固定值 0x1；
- *Type* 类型，4bit 长，固定值 0x1；
- *Code* 码值，8bit 长，PPPoE 是客户端和服务器在多点链路上建立的一种会话 *Session*（会话表示通信双方要遵从某种交互机制建立、维持、终止关系），PPPoE 使用不同的 *code* 表示该数据帧处于 PPPoE 不同的阶段（*discovery*、*session*、*terminate*），如客户端在链路上寻找服务器，称为 *discovery* 发现阶段，那么这个数据帧的数码是 0x09，会话阶段的 *code* 就变成 0x00；
- *Session ID* 会话标识，16bit 长，客户端和服务器在多点链路上，大部分情况多个客户端同时和同一个服务器建立会话，所以服务器为每个会话分配一个会话标识进行区分；
- *Payload length* 载荷长度，16bit 长，用于指明 PPP 封装中的“*protocol*”和“*information*”的字节数；
- *protocol* 协议，即 PPP 封装中的 *Protocol* 字段，定义和图 2-11 一致；
- *information* 信息，即 PPP 封装中的 *Information* 字段，定义也和图 2-11 一致。

以太网、802.3 对应用数据有最小长度要求，如以太网要求数据最小为 46 字节，而 PPP 对信息却没有最小长度要求，所以当 PPP 信息字段长度小于 38 字节时（还有 2 字节协议，6 字节会话，合起来正好 46 字节），要求使用 *PAD* 以满足以太网最小帧长要求。以太网、802.3、802.11 也有最大帧长要求，如以太网数据部分最大为 1500 字节，那么 PPPoE 的信息字段就不能超过 1492 字节。

PPPoE 将会是本文介绍的第一个交互式协议，本文介绍协议主要靠分析交互过程中的数据帧，Stevens 在写原稿时使用的是 Unix 下的 tcpdump 工具进行捕获数据帧，然后通过画图和文字进行讲解。PC 发展至今，在 Windows 操作系统使用 Wireshark（第一章中介绍的开源多平台网络分析工具）配合 WinPcap 捕获数据帧，在 Linux 下依然使用大名鼎鼎的 tcpdump 捕帧，但分析还是可以借助 Wireshark 中 Statistics 统计自动化工具中 Flow Graph 的帮忙，它有自动化图形分析结果，这么做的好处是直观，省去制图时间，也有利于大家亲自操作（Wireshark 有比较详细的 guide，比较推荐掌握的是 *filter* 过滤），分析 PPPoE discovery 发现阶段交互如图所示：

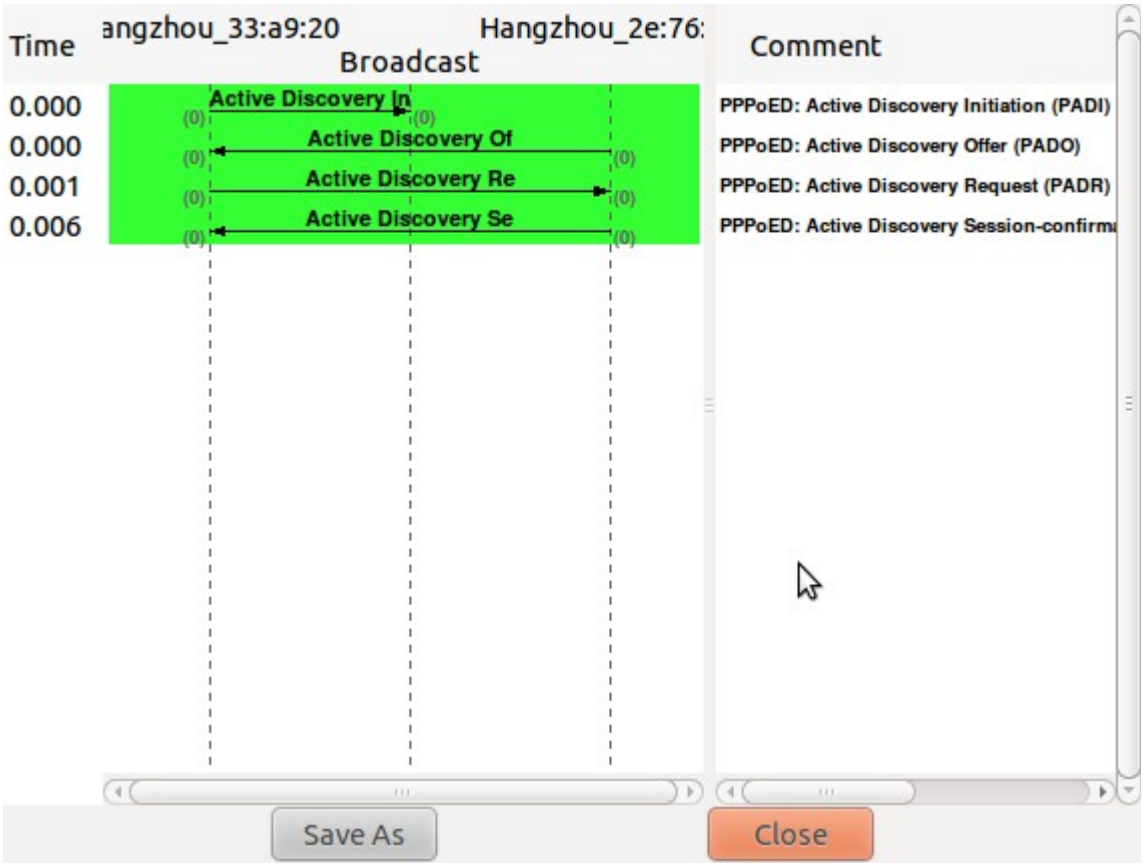


图 2-14 PPPoE Discovery 阶段图形分析（左为客户端、右为服务器）

PPPoE 完整通信过程共分为 Discovery 发现、Session 会话、Terminate 终止三个阶段，这里说完整是从分析角度来说的，实际使用过程中，会有可能因为人为因素（突然断电、线路故障）而变得不完整。正常的 Discovery 过程一共是 4 次数据帧交互：

```
▶ Frame 1 (60 bytes on wire, 60 bytes captured)
▼ Ethernet II, Src: Hangzhou_33:a9:20 (00:23:89:33:a9:20), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Source: Hangzhou_33:a9:20 (00:23:89:33:a9:20)
    Type: PPPoE Discovery (0x8863)
▼ PPP-over-Ethernet Discovery
    0001 .... = Version: 1
    .... 0001 = Type: 1
    Code: Active Discovery Initiation (PADI) (0x09)
    Session ID: 0x0000
    Payload Length: 10
  ▼ PPPoE Tags
      Host-Uniq: 000A
```

```
0000  ff ff ff ff ff ff 00 23  89 33 a9 20 88 63 11 09   .......# .3. .c..
0010  00 00 00 0a 01 01 00 00  01 03 00 02 00 0a 00 00   ........ ........
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00              ........ ....
```

图 2-15 PPPoE 发现帧 1（客户端发送）——*Initiation* 初始化

从 Wireshark 可以看到发现帧 1 完整的以太网封装结构：

- Destination MAC 是一个称为 *Broadcast* 的地址 ff:ff:ff:ff:ff:ff，这种地址的中文译名叫“广播”，顾名思义，以太网链路上所有的节点都要接收处理，它属于链路上的所有节点，只属于链路上某个节点的地址称为 *Unicast*“单播”地址，而属于链路中某一部分节点的地址称为 *Multicast*“组播”地址，根据以太网目的地址类型，可以把以太网帧分成广播帧、单播帧、组播帧；
- Source MAC 是 00:23:89:33:a9:20，属于图 1.12 中 Ethernet 客户端 1；
- Type 字段是 0x8863，表明以太网封装的是 PPPoE discovery 或 terminate 阶段；
- PPPoED 封装中关键的内容是 Code（0x09 表明是 Initiation 消息）
- Session ID 为 0，即没有设置，其值需要 PPPoE 服务器指定；
- PPPoE Tags 封装的信息是 *Host-Uniq*，这是个长度变化的字段，用来表明客户端身份，这里长度是 2 字节，身份是 000a，身份的填写依据于不同客户端的实现，如 Windows 会使用 12 字节身份；
- Wireshark 提示这个帧长度是 60 字节，而整个以太网帧中有用的信息包括 6 字节目的地址、6 字节源地址、2 字节 Type、6 字节 PPPoE 头、10 字节 PPPoE 载荷，总计 30 字节，30 字节长度差距就是以太网最小载荷长度引起的，以太网最小载荷长度是 46 字节，而在这里以太网载荷是 6 字节 PPPoE 头和 10 字节 PPPoE 载荷，离最小载荷长度正好 30 字节，故这里可以看到以太网填充 *Padding* 有 30 字节，需要注意的是，有的操作系统在抓取某些数据帧时会将主动将填充剥离，Wireshark 显示帧长不足 60 字节。

第 2 个数据帧是 *Offer* 消息：

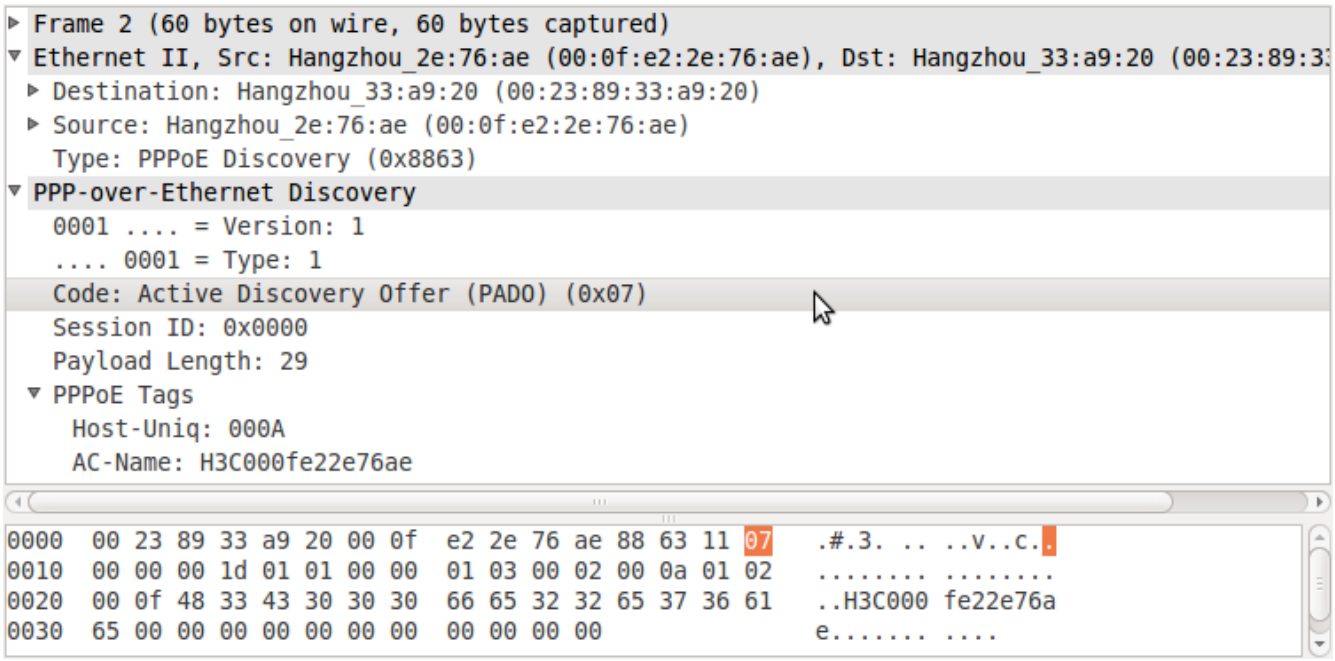


图 2-16 PPPoE 发现帧 2（服务器发送）——*Offer*

Offer 这个词不太好翻译，用处却广，出国留学、找工作也有 Offer，因为发音简单，就不译了。Offer 消息是个单播帧：

- Destination 地址，是客户端 MAC 地址 00:23:89:33:a9:20；
- Source 地址，这里是服务器 MAC 地址 00:0f:e2:2e:76:ae，Wireshark 显示 Hangzhou_33:a9:20 和 Hangzhou_2e:76:ae 并不奇怪，以太网 MAC 地址前 24 位是厂家标识 *OUI*（Organizationally Unique Identifier），由以太网接口制造商掏钱向 IEEE 购买，地址后 24 位则由厂家自行分配，这种机制可以确保以太网 MAC 地址不会冲突，每个厂家购买了哪些 OUI 可以从 IEEE 官方网站查询，如果像偷点懒的话也可以从 Wireshark 安装目录的 *manuf* 文件查看，经过查询 00:0f:e2 和 00:23:89 属于 HangzhouH3，所以 Wireshark 显示的时候直接将 OUI 用厂家名称代替，有的厂家名称太长，所以它只取前 8 位字符，由于厂家经常会合并，所以 OUI 值不变，厂家名称倒是有可能不停地刷新；
- PPPoE 中的 Code 是 0x07，表明这是个 *Offer*，Initiation 消息的 Code 还记得吗？
- Session ID 依然是全 0，因为此时 PPPoE 会话并没有建立，只是 PPPoE 客户端和服务器相互发现了对方而已；
- 载荷长度变成了 29 字节，和 Initiation 相比，多了 AC-Name 字段，AC 是 *Access Concentrator* 接入汇聚器的意思，也就是 PPPoE 服务器，里面填写内容是 15 字节长，AC 厂家实现格式是设备名称 H3C 和 MAC 地址字符的组合。

第 3 个数据帧是 *Request* 请求消息，意思是客户端请求服务器建立会话：

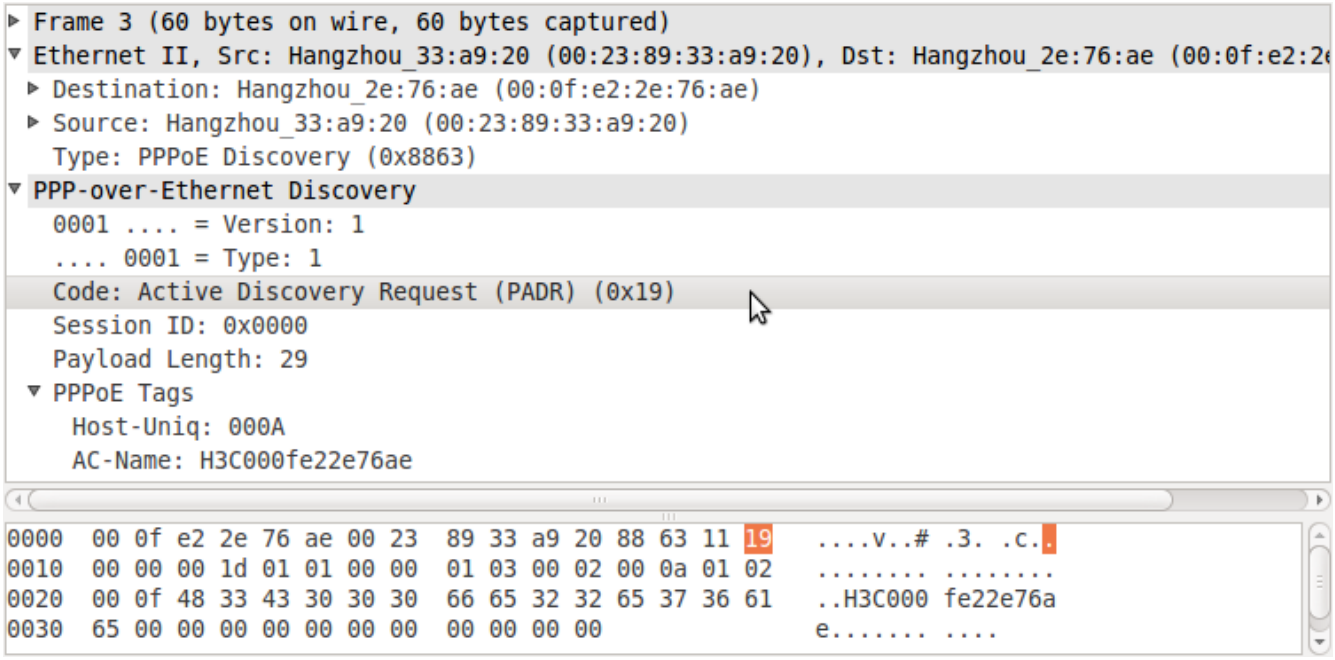


图 2-17 PPPoE 发现帧 3（客户端发送）——*Request* 请求

- 从帧 Destination 和 Source 地址分析，这是客户端发出的单播帧；
- 该帧除了 Destination、Source 和 Offer 帧正好相反（称为镜像地址对），Code 字段也不一样，是

0x19，这个值表明这个帧是由客户端发出的 Request 帧；

- 其余字段包括帧长都和 Offer 帧一致。

第 4 帧，也就是最后一个帧，称为 *Session-confirmation* 会话确认消息：

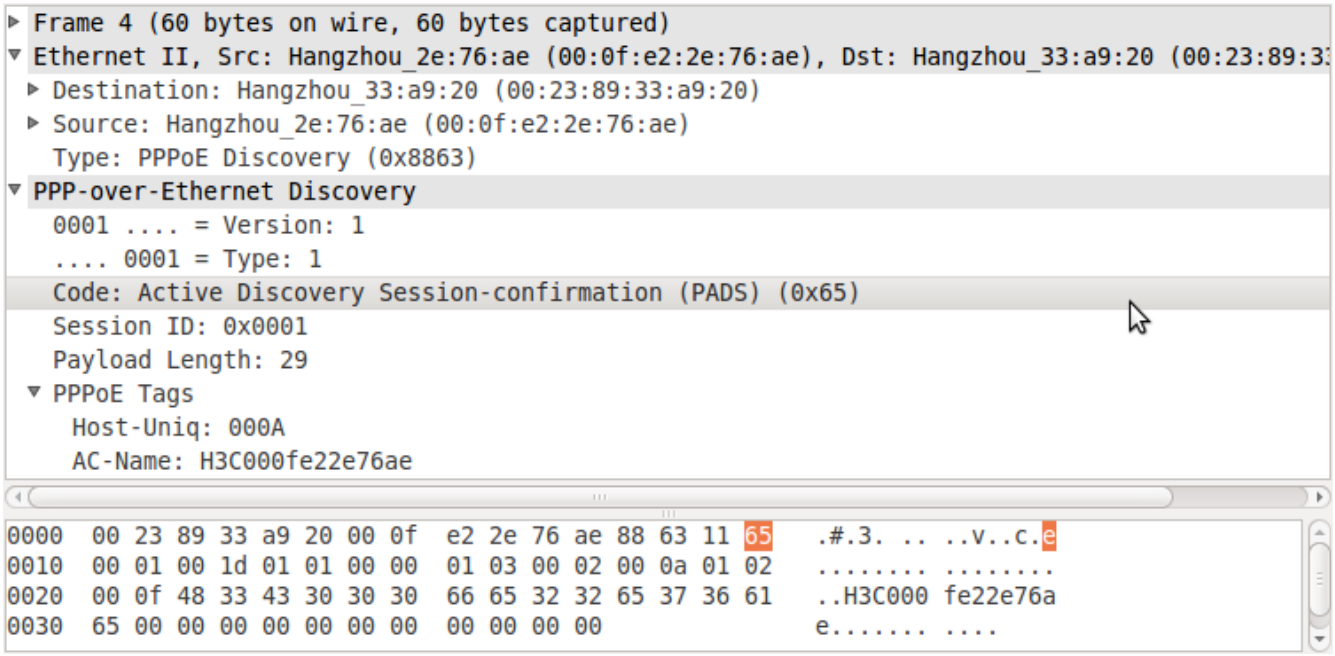


图 2-18 PPPoE 发现帧 4（服务器发送）——*Session-confirmation* 会话确认

- 会话确认的标志就是 Session-ID 不再是 0，而是一个数值，这里是 0x0001，服务器厂家在这里是采取顺序取值的实现方式；
- 会话确认消息的 Code 是 0x65；
- 其余字段、帧长与 Offer 帧一致。

从图 2-14 还可以看到 4 个数据帧分别是在什么时间捕获的，时间是相对时间，第 1 帧作为时间 0 点，可以从时间上看到 PPPoE 会话建立过程在 0.006s，也就是 6 毫秒内完成了，当客户端收到会话确认消息后，说明 PPPoE 会话已经建立，可以进行 PPP 链路活动了，在 2.8 节中介绍到了 PPP 是一个需要 LCP、身份认证、NCP 协商的链路层协议：

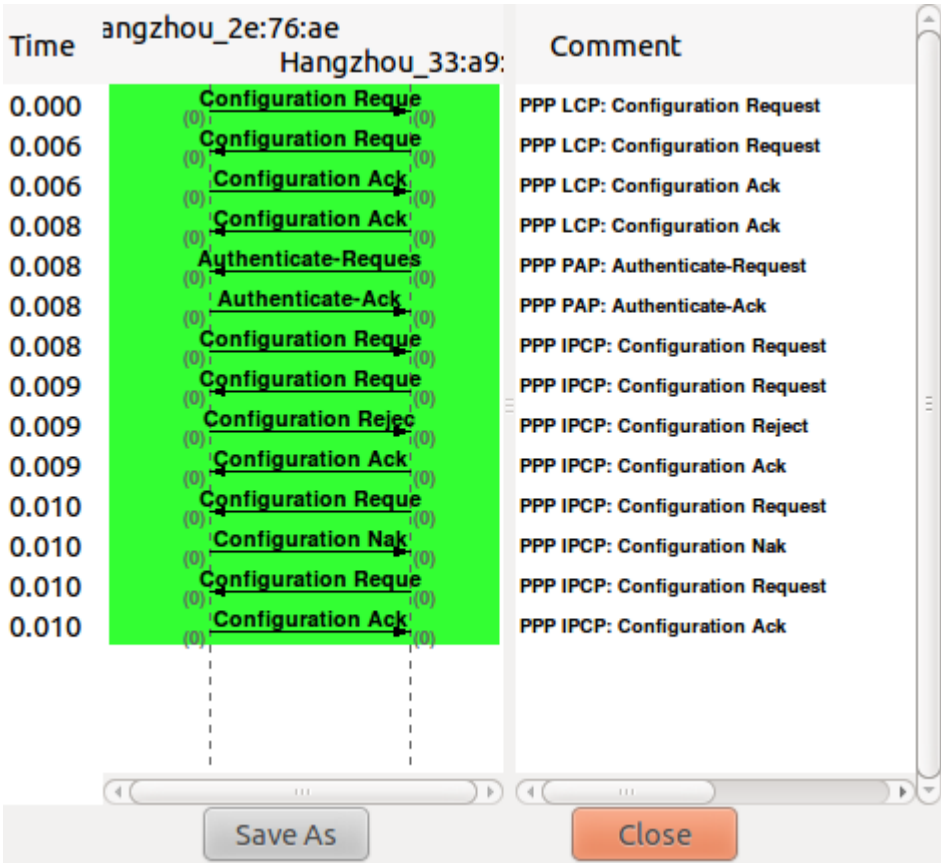


图 2-19 PPP 会话协商流程图（左为服务器，右为客户端）

从图 2-19 可以看出 LCP 协商总共涉及 4 条消息，这 4 条消息包括：

- 服务器向客户端发送 *Configuration Request* 配置请求，里面包括 MRU（*Maximum Receive Unit* 最大接收单元）、认证方式等协商内容；
- 客户端向服务器发送配置请求，协商内容与第 1 条消息一样；
- 服务器向客户端回应 *Configuration Ack* 配置确认，对客户端配置请求中的 MRU、认证方式表示同意；
- 如果不支持可以通过 *Configuration Reject* 配置拒绝应答，对具体的配置项进行拒绝；
- 如果是支持该选项但是不支持该数值，可以使用 *Configuration Nak* 应答，Nak 中携带所支持的数值，这是不是觉得和谈判很像，网络是人设计的，协议也就是人化的代码；
- 客户端向服务器发送配置确认，对服务器请求进行确认，内容也与第 3 条消息一致。

PPP 协商的内容都是单向的，比如服务器向客户端请求能力 A，客户端回以 Ack，说明服务器使用能力 A 时，客户端可以正确处理，并不意味着客户端使用能力 A，服务器也可以正确处理。也就是说是如果客户端要使用能力 A，也必须向服务器请求一下，只有服务器回 Ack，客户端方能使用。可以从中了解到协议的严格性，以及协议设计者的严谨。

由于消息比较多，许多消息格式差别不大，所以这里取服务器发送请求、客户端确认帧进行分析：

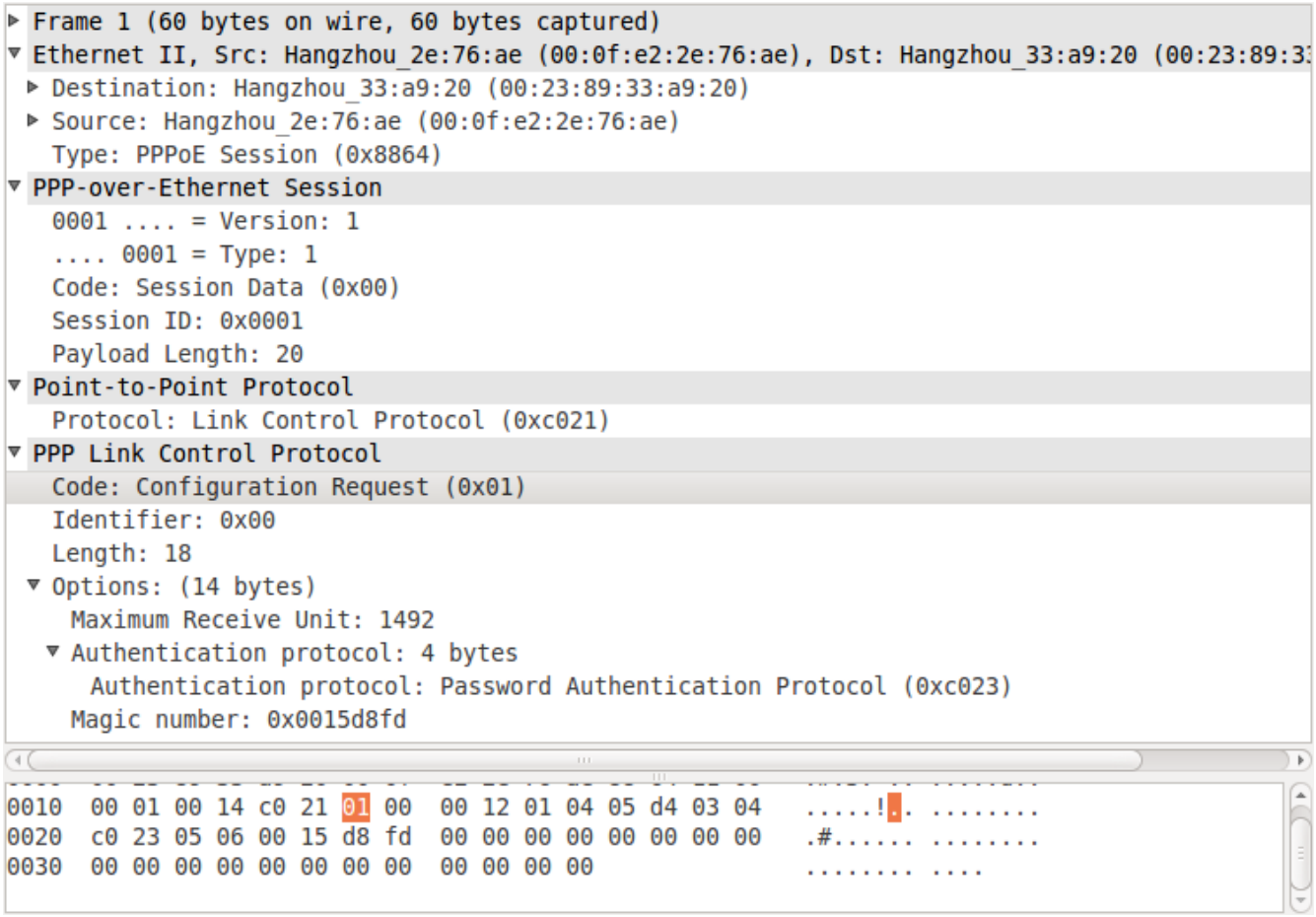


图 2-20 LCP 配置请求（服务器发送）

- 以太网 Type 字段值是 0x8864，表示封装的是 PPPoE 会话，还记得 PPPoE 发现的 Type 值吗?
- PPPoE 中的 Code 是 0x00，PPPoE 会话阶段都是这个值，表示会话数据 *Session data*;
- *Session ID* 就是会话发现最后阶段确定的 0x0001，同一个会话的数据都会使用相同的 Session ID;
- *Payload Length* 是 20 字节，再加上 PPPoE 头 6 个字节，以太网载荷长度为 26 字节，不足 46 字节，所以在这个帧中有 20 字节的填充;
- 接下来就可以看到 PPP 封装了，以往常见的 PPP 都运行在电话线、E1 专线、串行线路，捕获数据帧比较困难，而 PPPoE 是在以太网上运行，以太网上捕获数据帧的方法比较多，如通过一个共享介质的 Hub，或者是通过以太网交换机端口镜像等功能都能够捕获链路上通信的所有帧，分析数据帧是了解网络运行的良好手段；PPPoE 中的 PPP 是从 Protocol 字段开始的，flag、地址、控制三个字段都被略去，0xc021 表明这是 LCP 消息;
- LCP 消息中也有 *Code*，0x01 表示这是 C*onfiguration Request*，0x02 表示这是 *Configuration Ack*;
- *Identifier* 字段在需要对交互信息进行标识时使用;
- *Length* 字段，说明整个 LCP 消息的长度，这和常见的 *Payload Length* 不太一样，长度说明是包含

Header 在内的整个消息长度，而载荷长度这是指除 *Header* 后封装数据部分长度，这里可以看到 LCP 封装的 *Options* 长度为 14，算上 *Code*、*Identifier*、*Length* 为 18；

- 第 1 个 *Option* 选项是 MRU，意思是这条链路上能接收的 PPP 封装数据部分最大长度，MTU 则表示发送方向最大长度，通常一条链路上的 MRU ≥ MTU，MTU 是由链路标准化组织指定的，而 MRU 则和具体的链路层实现有关系，不同的厂家实现 MRU 的默认值有可能不一样，甚至是可以变化的，所以 MRU 是一个可以协商的内容，协商结果是双方 MRU 中的较小值，这里 MRU 为何是 1492 呢？从图 2-13 计算，以太网的 MTU 是 1500，刨去 PPPoE 会话 6 字节开销，PPP 协议字段 2 字节开销，最后就剩下 1492 了；
- *Authentication Protocol* 认证协议，即使用 *PAP*（密码认证协议）还是 *CHAP*（Challenge Handshake Authentication Protocol 挑战握手认证协议），或者不认证，在这里 0xc023 表示使用 PAP；
- *Magic Number* 魔术字，在 PPP 链路中，比较害怕链路成环，如图 2-9 所示，正常的情况下终端 1 的 Tx 连接终端 2 的 Rx，终端 1 的 Rx 连接终端 2 的 Tx，如果是终端的 Tx 和 Rx 连接在一起，就称 PPP 链路成环了（成环后，两个终端间将无法正常通信），这种成环通常都是特殊原因造成的（如人为线路连接错误，或者为判断链路故障故意打环），但 PPP 协议必须要有手段来判断点到点链路是不是成环了，魔术字通常是一个终端发送的随机数，如果终端接收到魔术字和发送魔术字一样，那么就可以基本判断链路成环了。

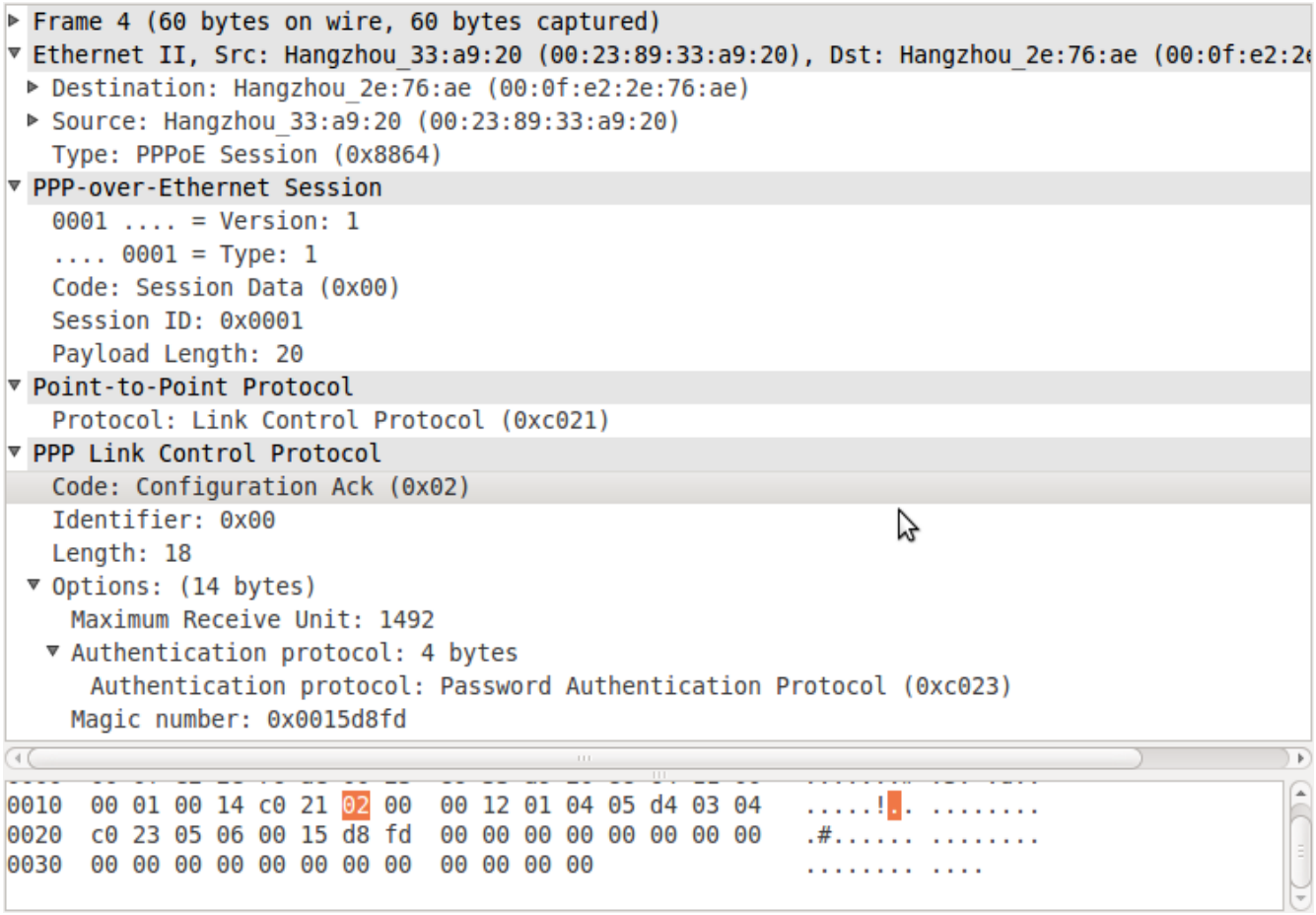


图 2-21 LCP 配置确认（客户端发送）

- 这个配置确认是客户端发送的，所以 Destination 和 Source 和服务器发送的请求正好相反，这种成对相反的数值，也被称为镜像数值；
- LCP 中的 Code 是 0x02，表示配置确认；
- 其余字段皆与配置请求相同，这里魔术字相同也问题不大，因为 PPPoE 是在以太网上传输，可以通过以太网源、目的地址判断是否真正成环。

前面提到了，在 LCP 协商中，同意的选项会通过 *Configuration Ack* 应答，不支持的选项会通过 *Configuration Reject* 配置拒绝应答，那么这个“拒绝”的模样如何，且来看看（从 MAC 地址可以知道这个数据帧和之前数据帧并不相同，是我从另外一个实验中抓取的，从其余实验中抓取的会有*标识）：

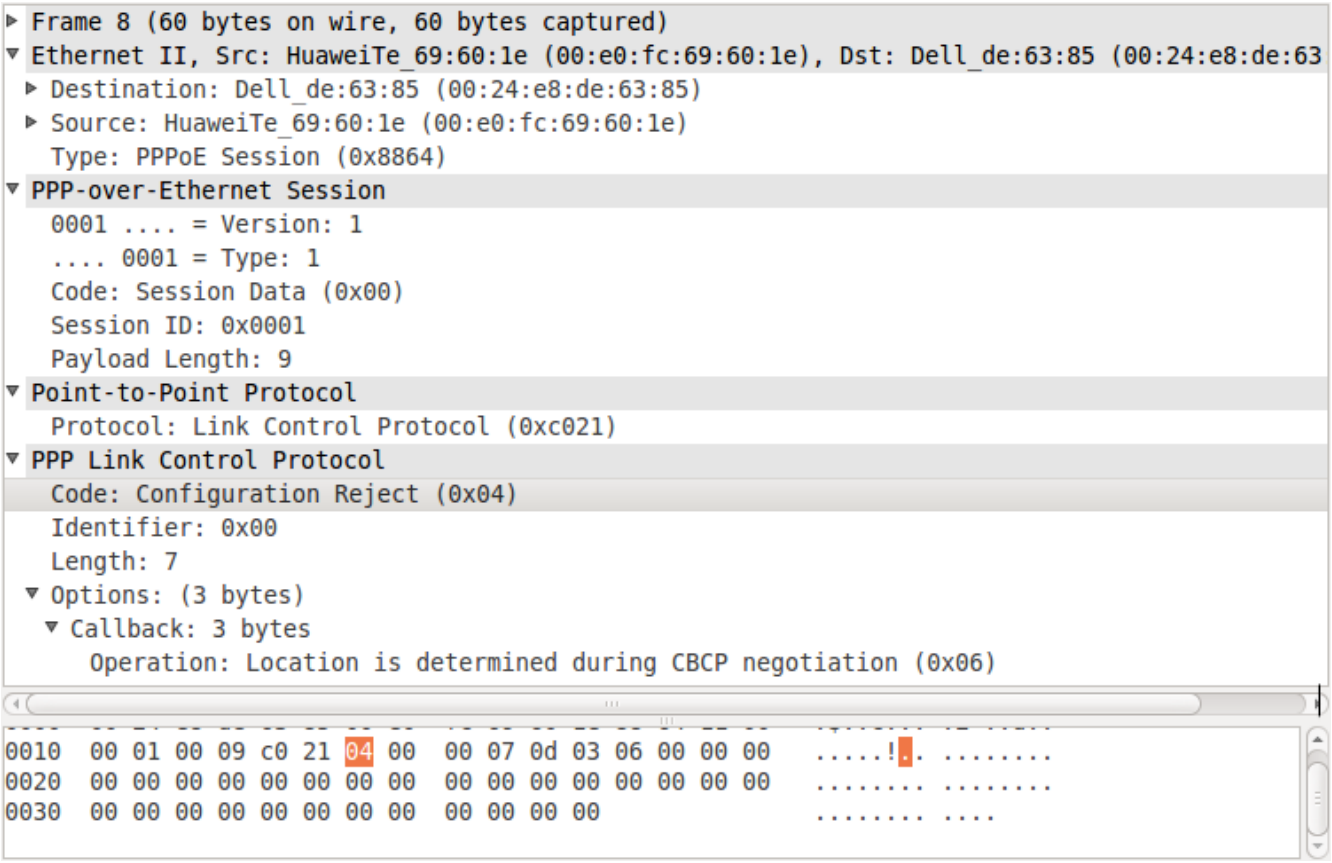


图 2-22 LCP 配置拒绝（*服务器发送）

- LCP 中的 Code 值为 0x04，意思就是本消息是 *configuration Reject*；
- 具体哪个选项因为不支持，而被拒绝呢，*Options* 里头显示了，是叫 *Callback* 回呼；
- 可见拒绝和同意的选项都是在 *Options* 显式确认，并不是隐式的。

下面再来看 PPP 认证过程中的 *PAP* 认证：

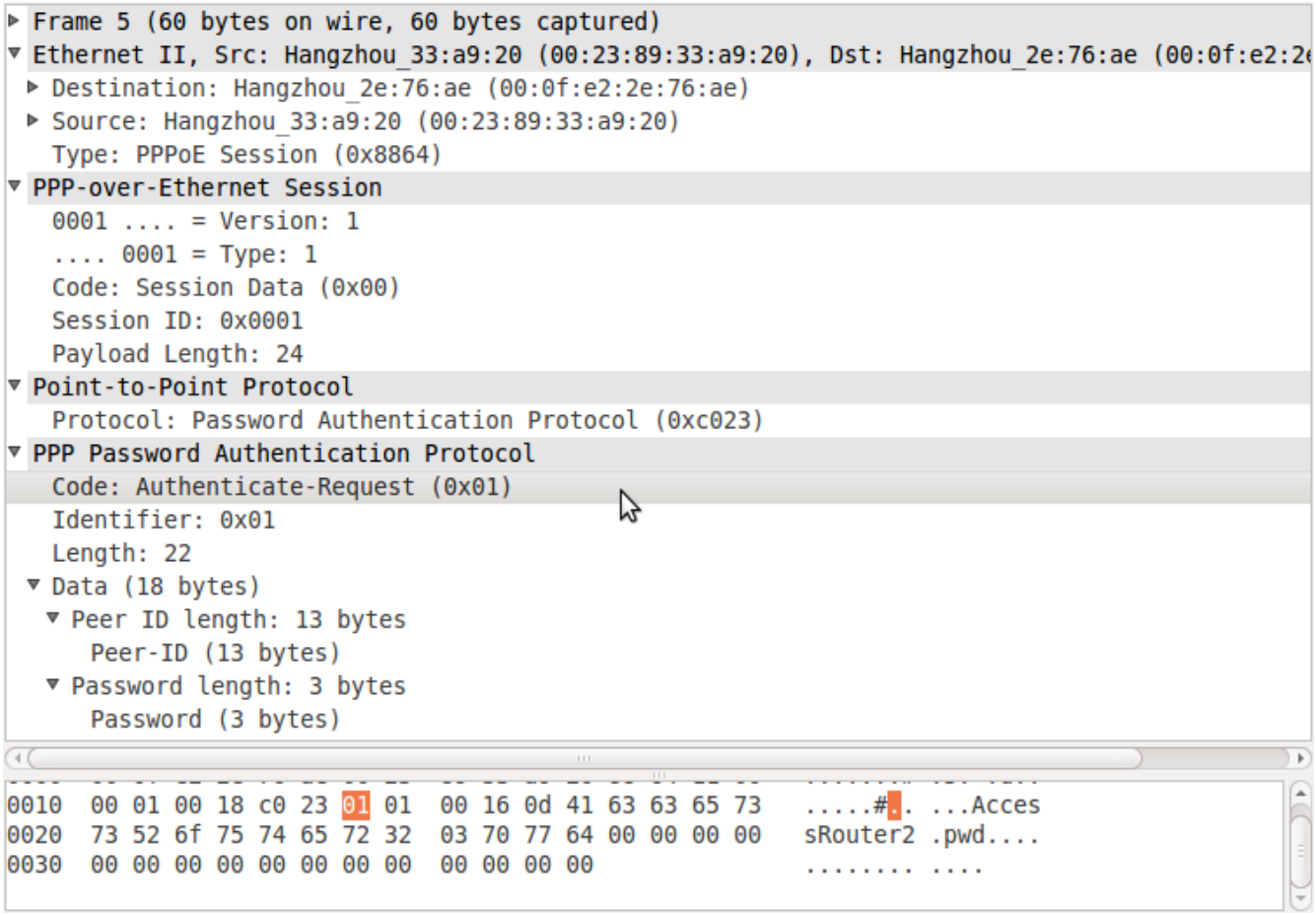


图 2-23 PAP 认证请求（客户端发送）

- 从以太网的 *Source*、*Destination* 地址看，这是客户端发给服务器的，*Type* 也是 0x8864，PPPoE 的 *Code*、*Session ID* 和之前 LCP 是一致的；
- PPP 的协议字段是 0xc023，说明是封装的是 *PAP* 协议；
- PAP 中的 *Code* 值是 0x01，说明是 *Authenticate-Request* 认证请求消息；
- *Identifier* 在这里也设置成 0x01，意思是只有 Identifier 也为 0x01 的 *Ack* 消息才是对此 *Request* 的回应；
- PPP 中封装的内容称为 *Data* 数据，里面的内容就是用于身份确认的用户名 *Peer-ID* 和密码 *Password*，可以看到用户名长 13 字节（AccessRouter2），密码 3 字节（pwd）。

如果这个用户名密码和服务器上配置的用户名密码一致，那么服务器就会给客户端回复 PAP 认证确认 *Authenticate Ack* 消息：

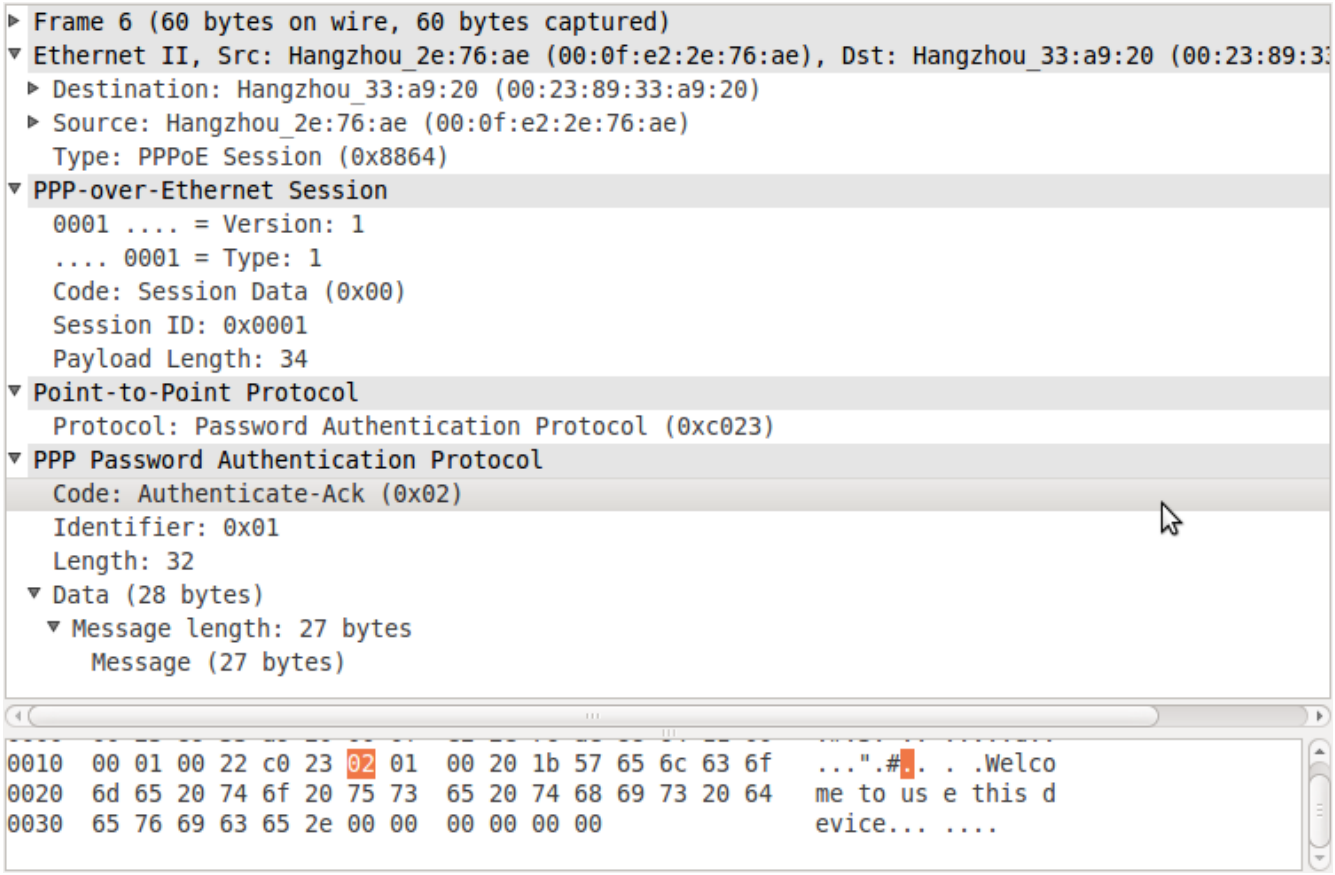


图 2-24 PAP 认证确认（服务器发送）

- 从 MAC 地址看，这个认证确认帧是服务器发给客户端的；
- 和认证请求相比，不同地方在于 PAP *Code* 变成 0x02，表示是认证确认 *Authenticate-Ack*；
- *Identifier* 也是 0x01，说明是应答 *Identifier* 也为 0x01 的认证请求消息；
- *Data* 部分变成 *Message*，这条消息内容比较正经“Welcome to use this device”，译为“欢迎使用这台设备”，如果换作是我，就会写成“How are you doing?”。

从图 2-19 中看，PAP 正常结束后就是 *IPCP* 阶段了，*IPCP* 分为 *Request*、*Ack*、*Rejec*、*Nak* 四种消息，从 IETF 查询 PPP 的 RFC1661 可以发现，里面并没有定义 IPCP 或者其余 NCP 协议，PPP 应该说是个协议群，这边定义 LCP、另外一个标准定义 PAP、CHAP、IPCP 等等，就像一个家族似的，一个家族通常会使用共同的特征，PPP 协议家族的特征就是协商时使用消息名称是一样的。下面就看 *IPCP Request* 消息的样子：

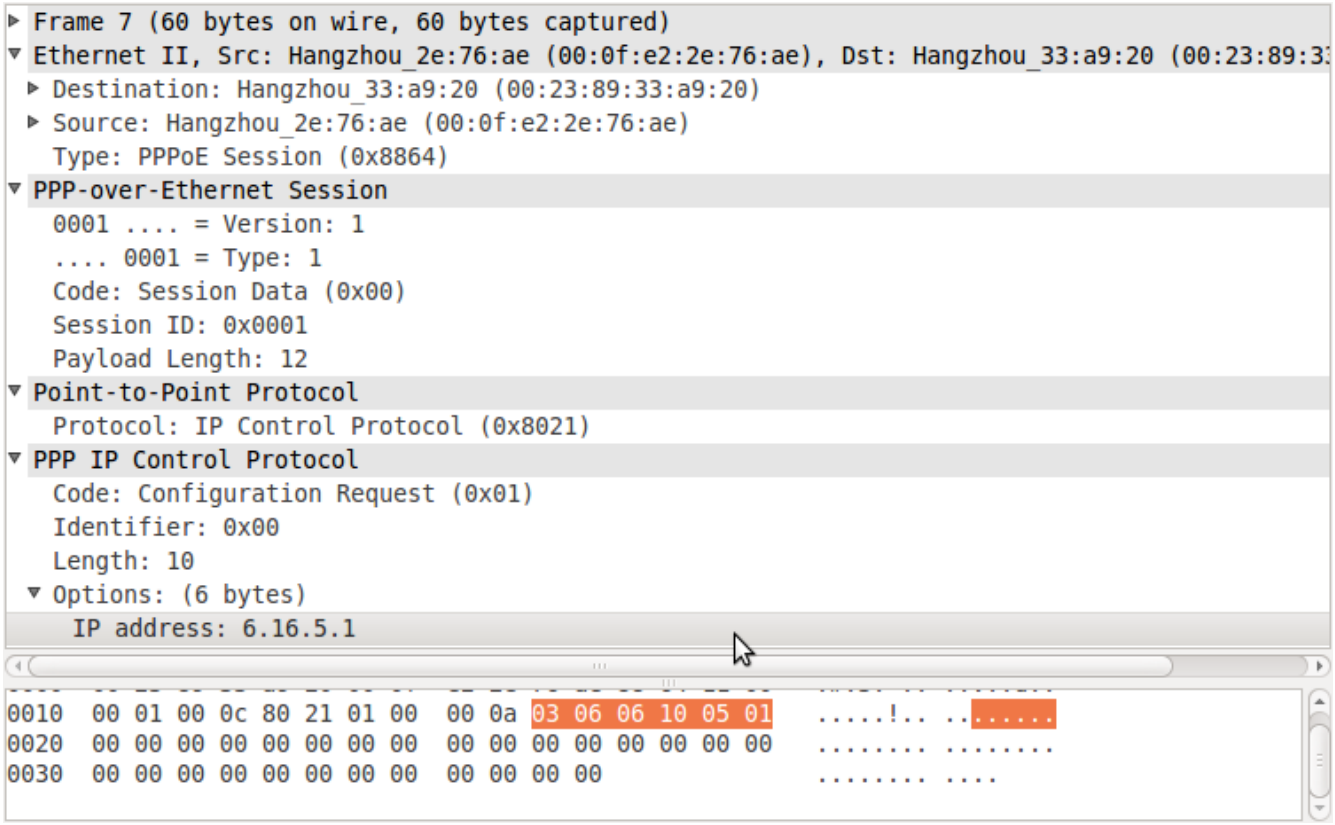


图 2-25 IPCP 配置请求（服务器发送）

- 从源、目的地址分析这是服务器发给客户端的，PPP 的 *Protocol* 是 0x8021，说明是 IPCP；
- IPCP 的 *Code* 是 0x01，说明是 *Configuration Request* 配置请求；
- 里面的 Options 是 *IP* 地址 *IP Address*，值是服务器侧的地址 6.16.5.1。

这个消息是服务器向客户端请求“服务器使用 6.16.5.1”，客户端意下如何，客户端也会向服务器发送请求：

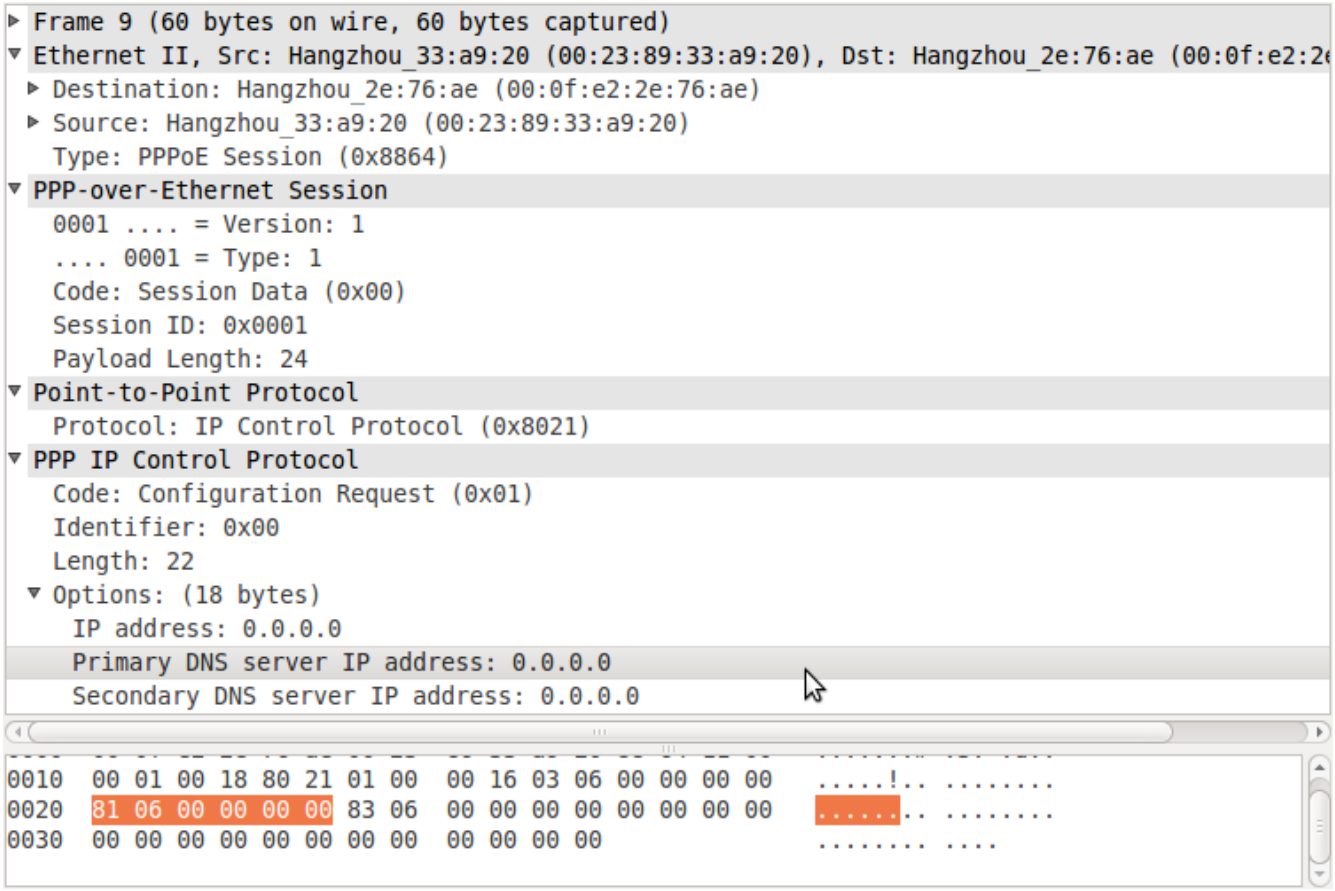


图 2-26 IPCP 配置请求（客户端发送）

重点观察 Options：

- 第 1 项也是 *IP Address*，但是值是 0.0.0.0，意思是要服务器分配；
- 第 2 项 *Primary DNS Server IP address* 主 DNS 服务器地址，值也是 0.0.0.0，意思也是需要服务器分配；
- 第 3 项 *Secondary DNS Server IP address* 从 DNS 服务器地址，值还是 0.0.0.0，意思也还是需要服务器分配。

服务器会怎么响应呢?

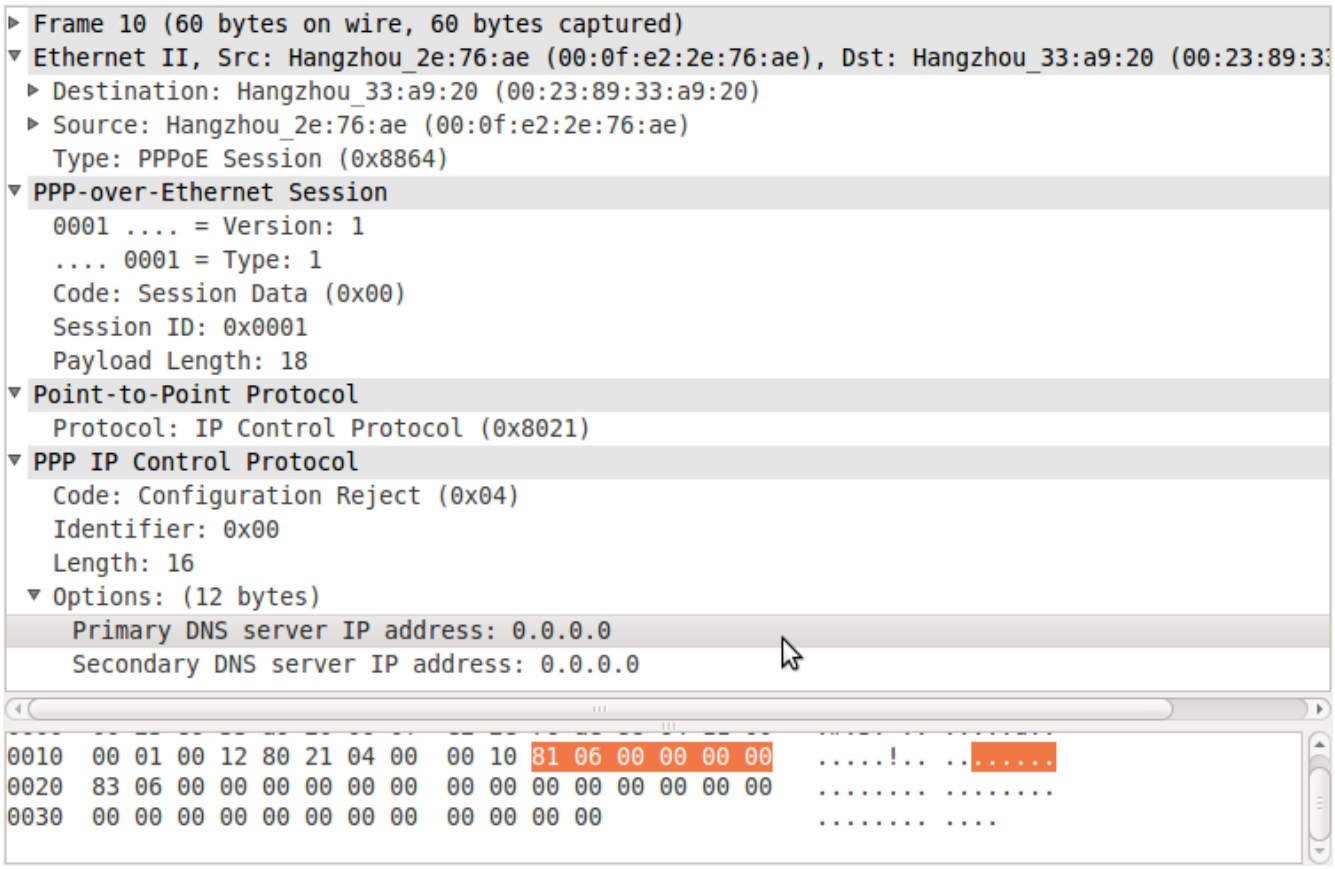


图 2-27 IPCP 配置拒绝（服务器发送）

服务器为什么拒绝客户端呢，仔细分析

- IPCP *Code* 是 0x04，表示 IPCP 配置拒绝 *Configuration Reject*；
- Options 里是主 DNS 服务器地址和从 DNS 服务器地址，这说明服务器不支持这两个选项。

那么客户端会如何应对这种尴尬的情形：

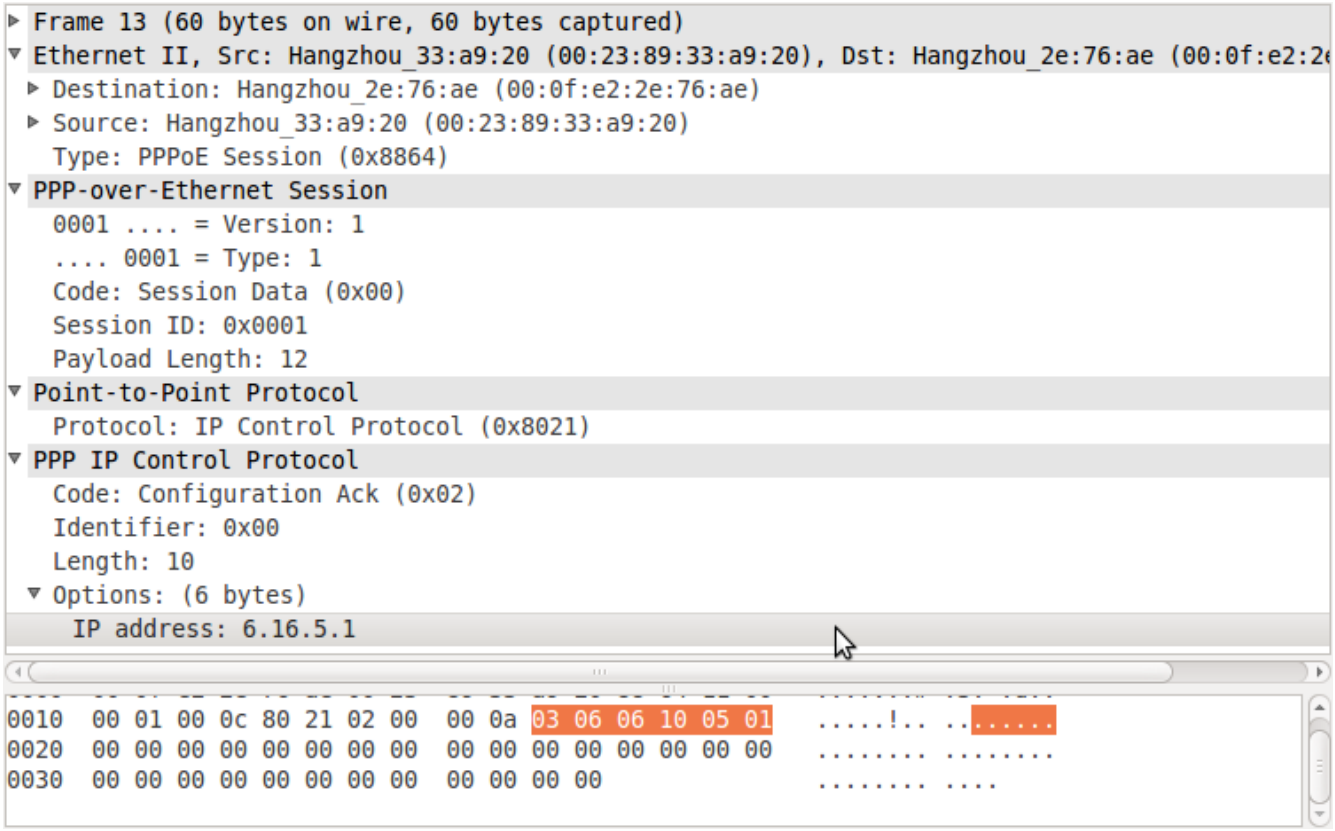


图 2-28 IPCP 配置确认（客户端发送）

- Code 是 0x02 表示 *Configuration Ack* 配置确认；
- 确认选项可以从 *Options* 中看出客户端同意服务器 IP 地址 6.16.5.1。

服务器拒绝了客户端的请求，客户端却对客户端请求以确认，有人认为客户端以德报怨，也有人认为客户端犯贱，其实都错了，客户端和服务器都是机器，只按照程序工作，确认只针对对端发送的请求，支持即确认，不支持即拒绝，不存在冤冤相报。服务器拒绝客户端请求，这说明客户端请求中 Options 有服务器不支持的，客户端重新请求就可以了：

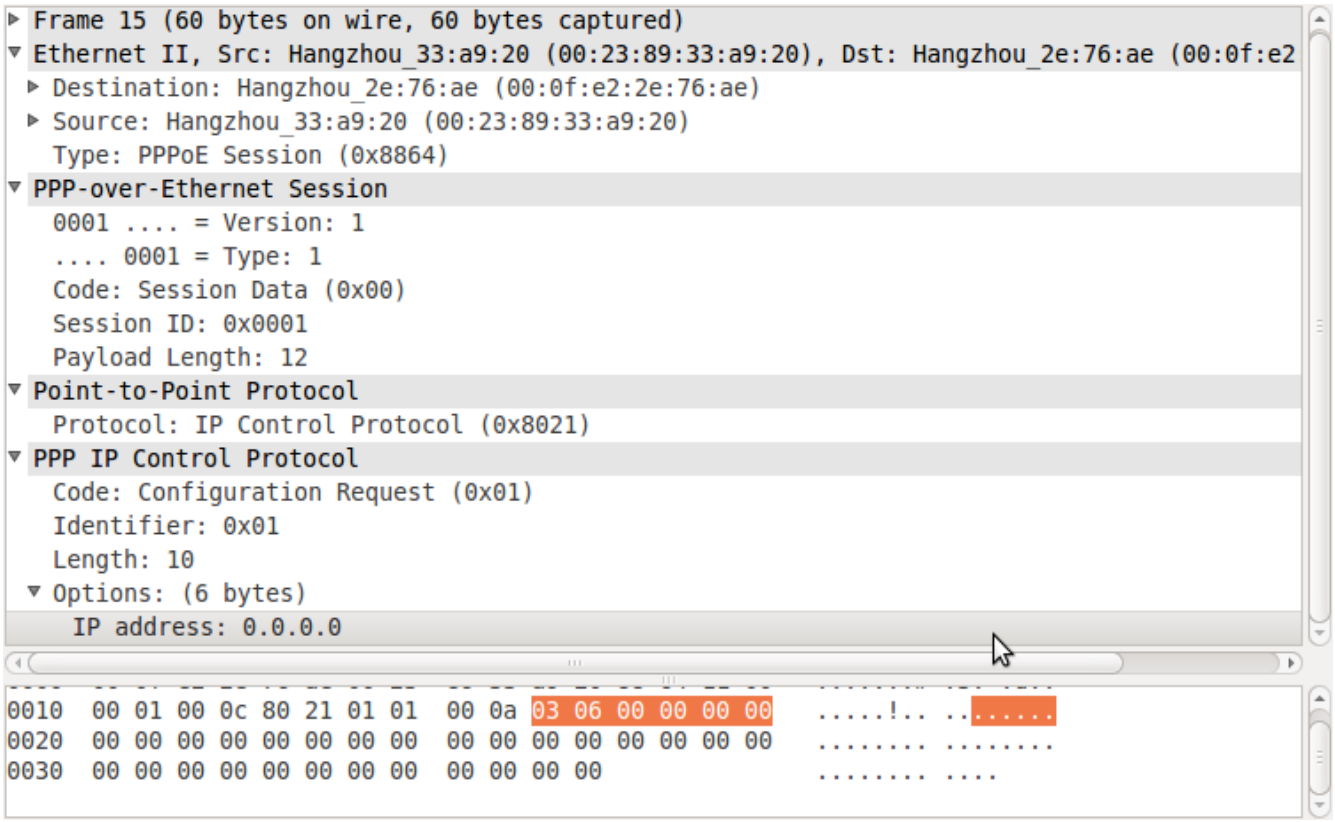


图 2-29 IPCP 配置请求（客户端发送）

- 对照图 2-26，客户端新发送配置请求中的 Options 可以发现，客户端把请求主 DNS 服务器地址、从 DNS 服务器地址 2 个选项取消了，客户端还是很友好的，服务器不支持，那么就不请求，换过来客户端如果不支持某个选项，服务器也会不请求的。

这次服务器应该会同意了吧：

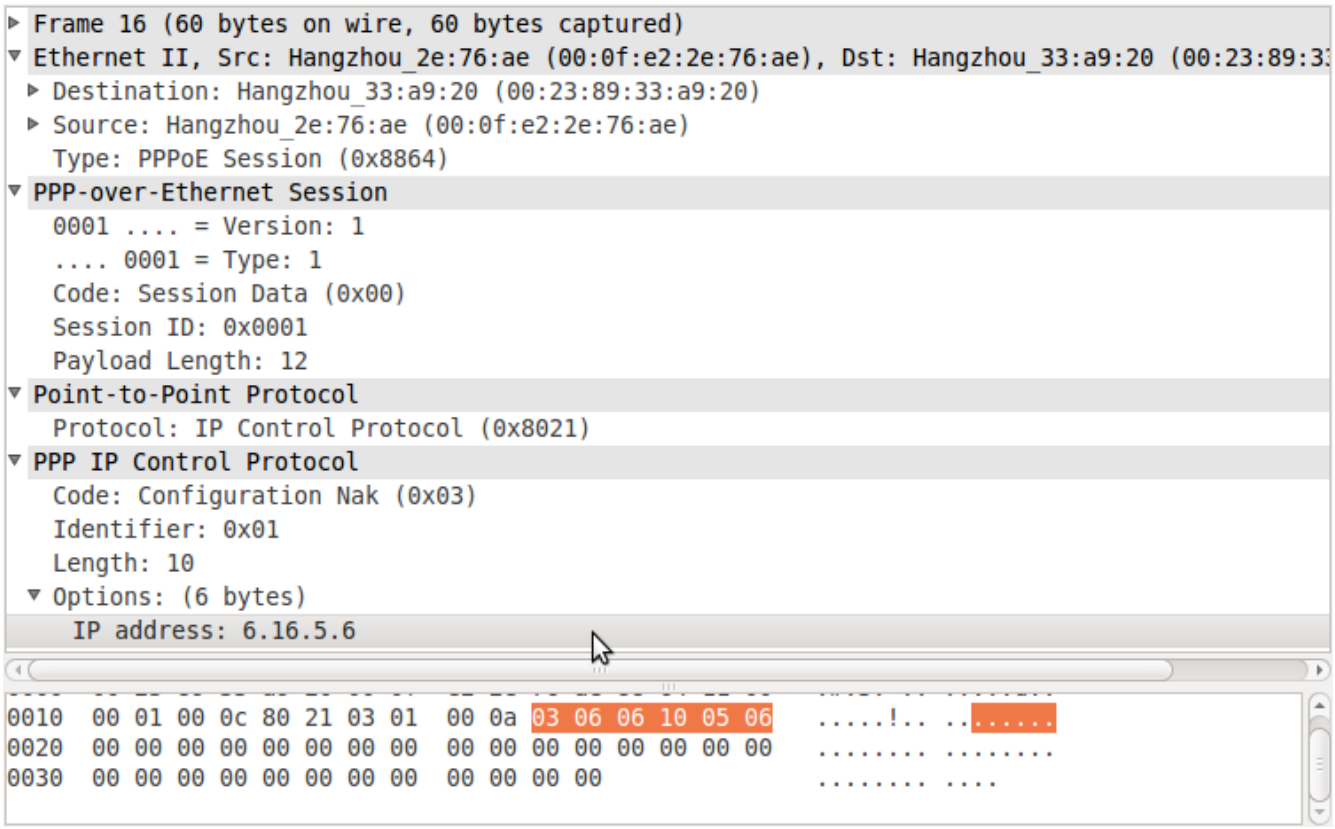


图 2-30 IPCP Configuration Nak（服务器发送）

- 大家可能会觉得服务器犯贱了，其实也不是，图 2-29 中客户端请求地址中写 0.0.0.0 也可以理解成客户端真的把 IP 地址设置成了 0.0.0.0，和服务器侧设置的客户端地址不一致，所以服务器会送 Nak，告诉客户端不能使用 0.0.0.0，应该使用 6.16.5.6。

客户端是没有收到 Ack 心不死，继续发送配置请求：

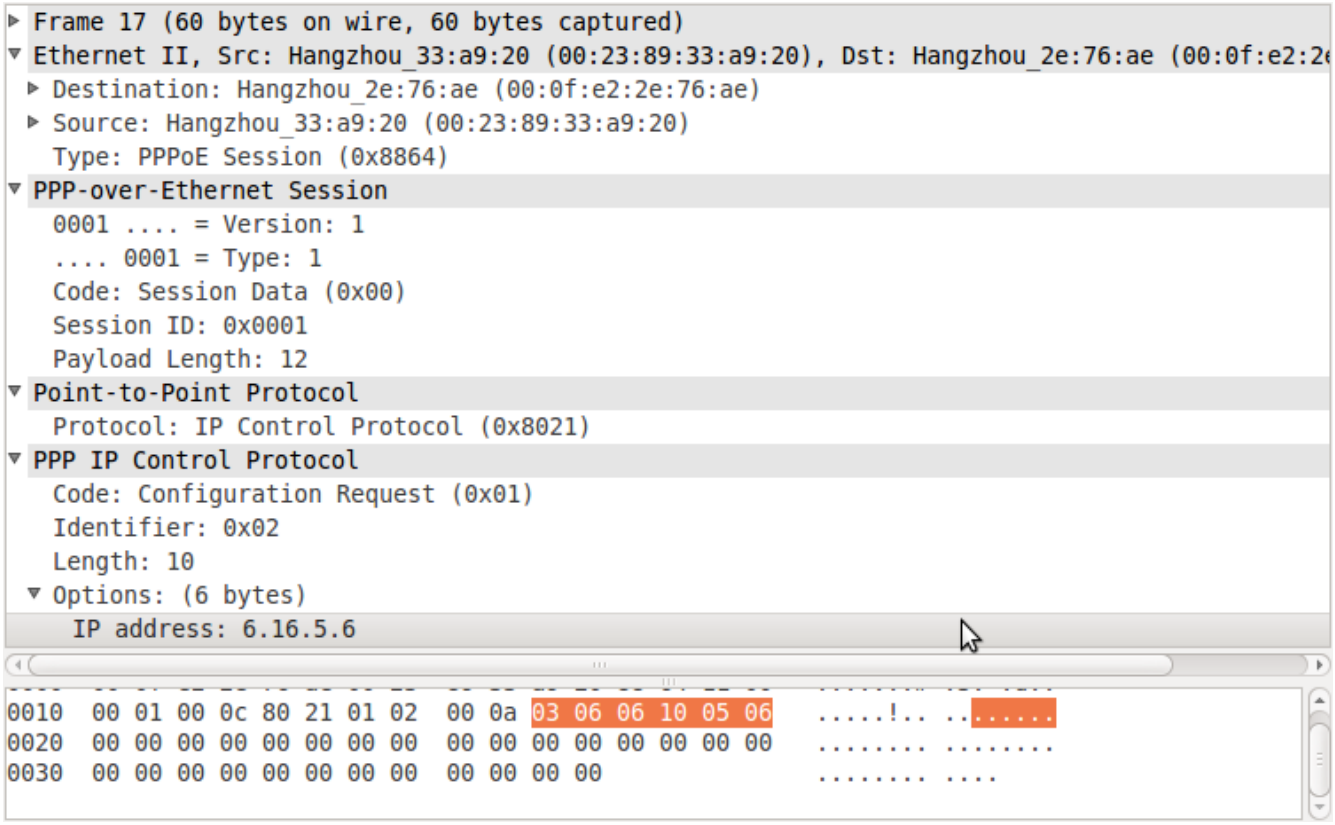


图 2-30 IPCP 配置请求（客户端发送）

- 客户端以服务器 Nak 消息中指定的 IP 地址 6.16.5.6 重新向服务器发起配置请求。

客户端如此仁至义尽，服务器是会确认的：

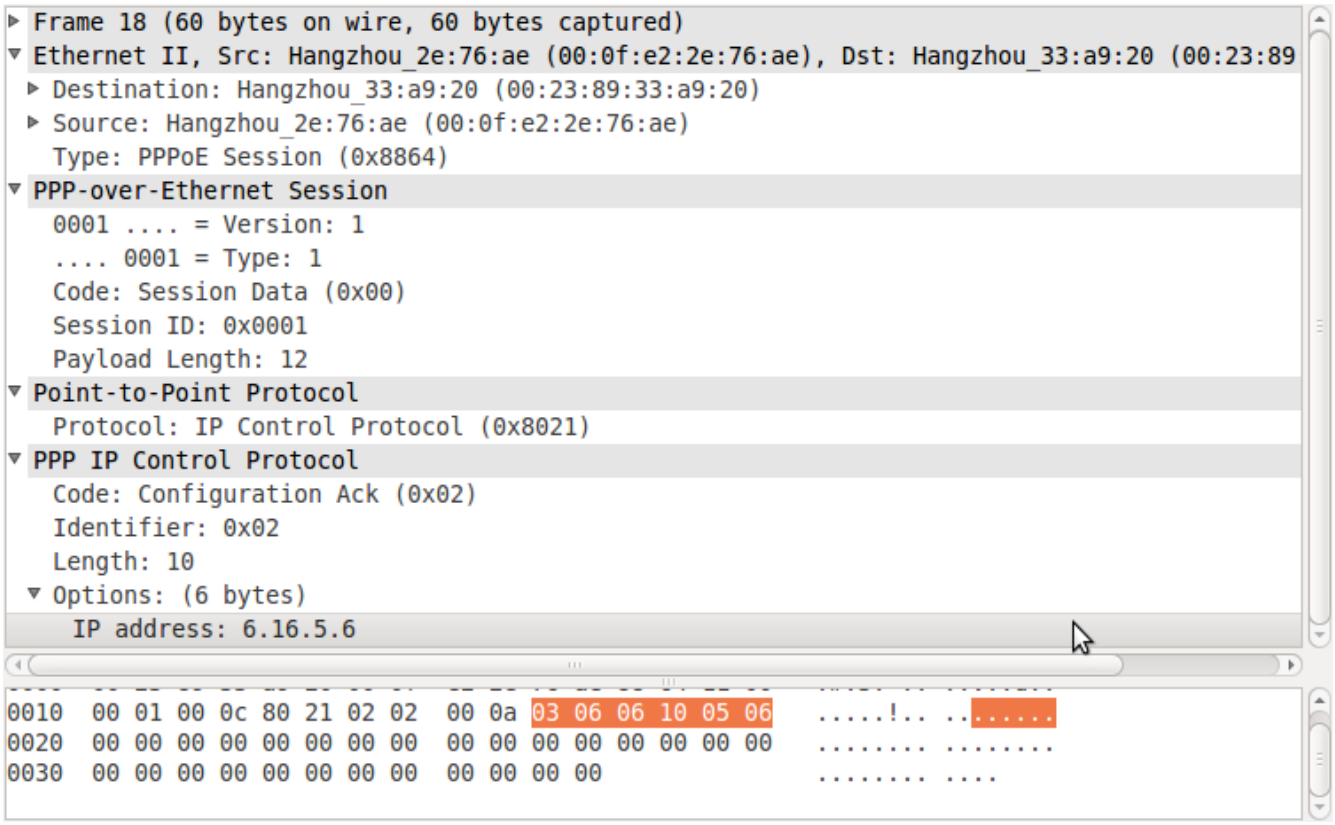


图 2-31 IPCP 配置确认（服务器发送）

总结一下，其实并不是服务器故意要折腾客户端，这一切都是协议的定义，协商结果以 Ack 作为达成共识的信号，收到任何 Reject 要把被拒绝的 Option 剔除后重新发送 Request 才能达成共识；收到任何 Nak，则要使用 Nak 中携带的数值重新发送 Request 达成共识；如果在一段时间内都没有收到 Ack 会怎么办，那么就是以协商失败而告终。

PPP 虽然是 TCP/IP 家族中比较基础、普通和简单的协议，但一介绍起来，每个细节都有很多可说的，之前介绍了 PPP 身份认证中的 PAP，另外一项 CHAP 认证也很流行：

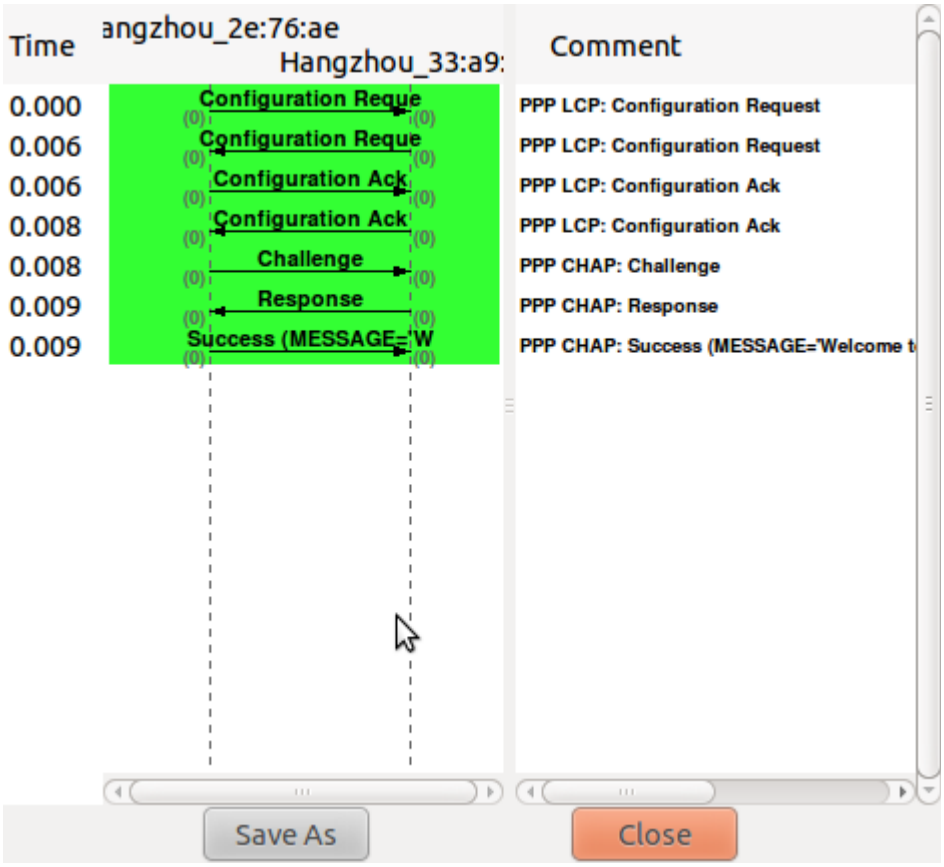


图 2-32 PPP CHAP 认证流程图（左为服务器，右为客户端）

这里把 LCP 部分协商流程也纳入进来是因为使用 CHAP 的 LCP 和使用 PAP 的 LCP 不一样，不一样在于服务器发送的 LCP 配置请求消息不一样：

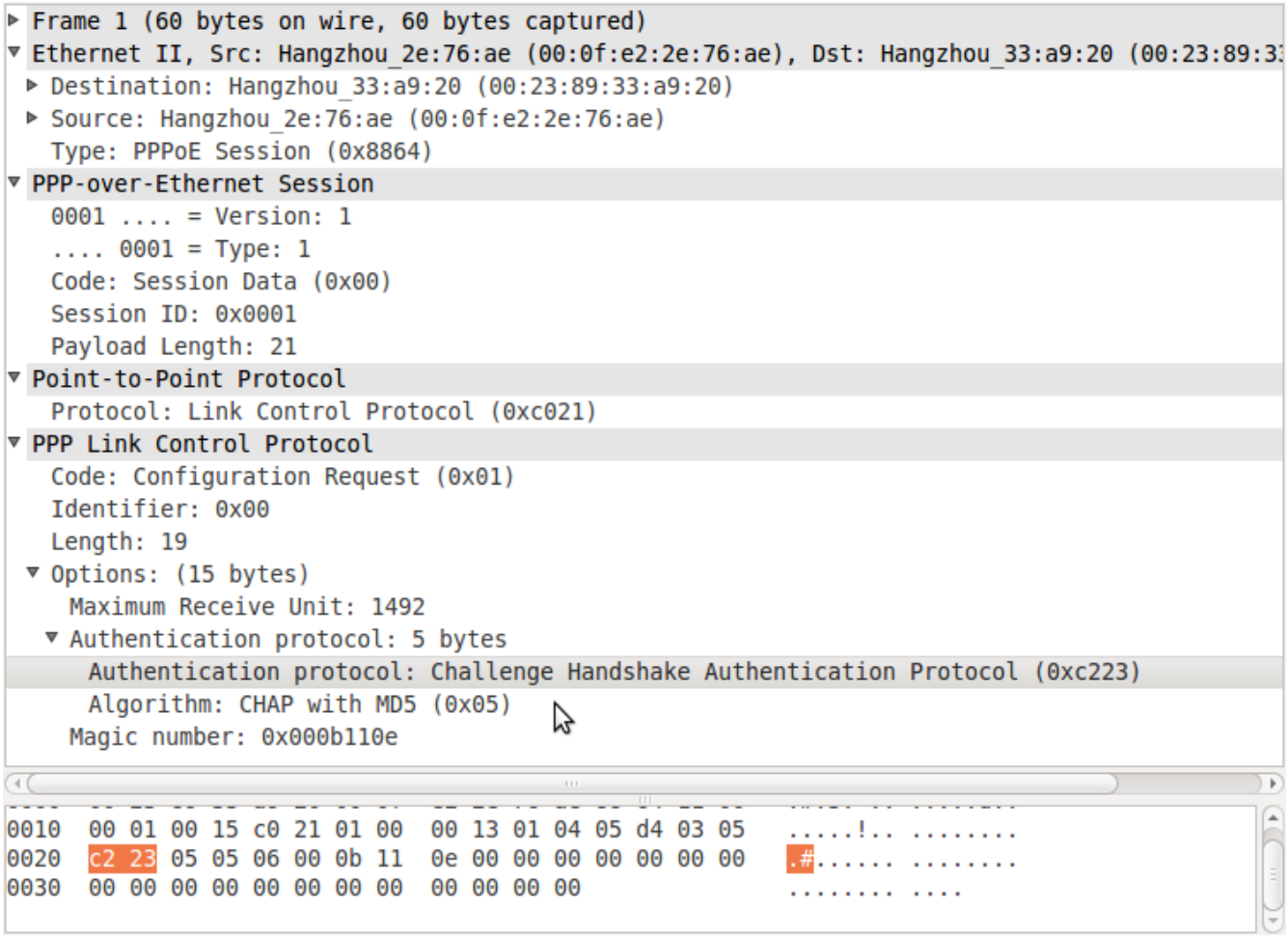


图 2-33 PPP CHAP 认证使用的 LCP 配置请求（服务器发送）

为什么选择服务器发送呢？因为在身份认证中，分为 *Authenticator* 认证方和 *Supplicant* 认证申请方。一般来说认证都是单向的，也就是服务器单向认证客户端，客户端不认证服务器，实际网络中也存在双向认证甚至三方交叉认证情况，但 PPP 链路中单向认证为主。

- 这个 LCP 中比较特殊的是 Options 中认证协议是 0xc023，表示使用 CHAP 作为认证协议；
- *Algorithm* 算法是 *MD5*（*Message Digestion 5*），一种被认为是很难破解的单向 Hash 函数，怎么理解呢，假设 MD5(How are you doing?) = 16bytes-String，也就是说使用 MD5 算法对“How are you doing?”这句话进行计算，结果是 16bytes-String；那么现在告诉你一句话使用 MD5 计算结果“16bytes-String”，你基本上无法推算出原话是“How are you doing?”，数学是很神奇的，我们要相信。

下面就来看一下认证过程：

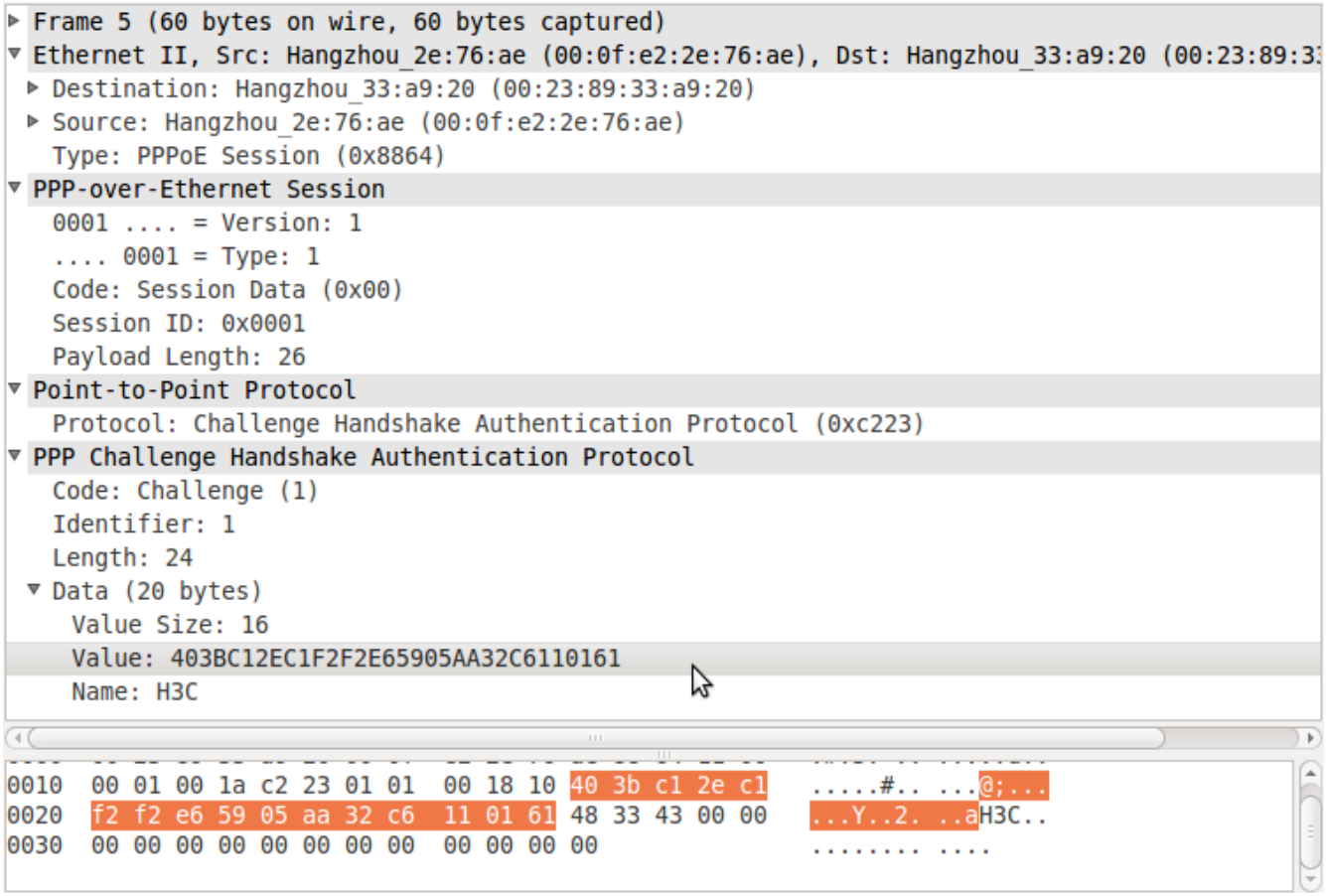


图 2-34 CHAP 的 *challenge* 挑战消息（认证方服务器发送）

- CHAP 封装中的 Code 值是 0x1 表示是挑战消息；
- Identifier 值是 0x1，在这里很有用处；
- 比较关键的是最后 Data 中 *Value* 数值部分，MD5 的计算结果是 128bit 的，也就是 16byte，这个数值是服务器随机生成的，并不是 MD5 的计算结果；
- *Name* 名字字段是表示认证方的名字。

那么客户端会怎么回应呢：

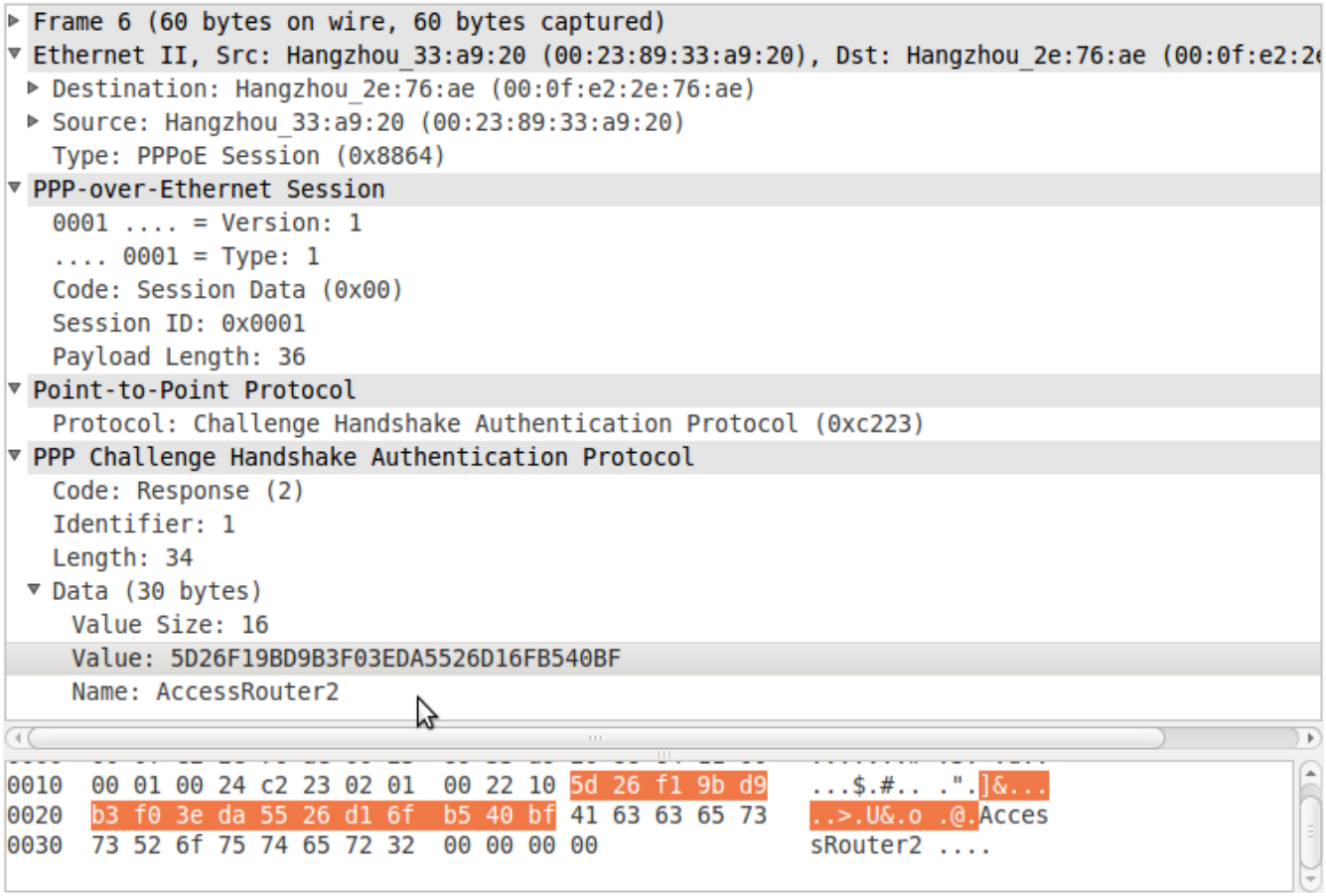


图 2-35 CHAP 的 *Response* 应答消息（被认证方客户端发送）

- CHAP 封装中 Code 是 0x2，表示这是应答消息；
- Value 部分也是 16 字节，这个是 MD5 计算结果，计算的素材是挑战消息中的 Identifier、用户身份认证信息（通常是用户密码）和随机数（挑战消息中的 Value）三者串联，这就保证了用户认证信息不会在线路上明文传递；
- Name 部分是被认证方的用户名 AccessRouter2。
- 那么服务器是怎么判断用户名和密码是否通过的呢?
  - 服务器发送的随机数（挑战消息中的 Value），这个随机数和 Challenge 消息中的 Identifier 存在对应关系；
  - 服务器作为认证方，必然保存了客户端的用户名和密码；
  - 服务器根据 Response 消息中的用户名找到对应的密码；
  - 服务器根据 Response 消息中的 Identifier 也是 0x1（Response 的 Identifier 和 Challenge 一致）找回刚才发送的随机数；
  - 服务器根据 Identifier、用户机密信息、随机数也用 MD5 计算数值，并和 Response 中的 Value（客户端的 MD5 计算结果）进行比对，如果一致就是认证通过，不一致就不通过，根据 MD5 单向算法的原理，任何人只知道随机数或者只知道密码都无法得出相同的 MD5 计算结果，

而密码又不以明文的方式出现在消息中（从 PAP 抓包可以看到客户端明文密码），从而保证了客户端身份信息的安全。

客户端认证通过后，服务器回应消息如下：

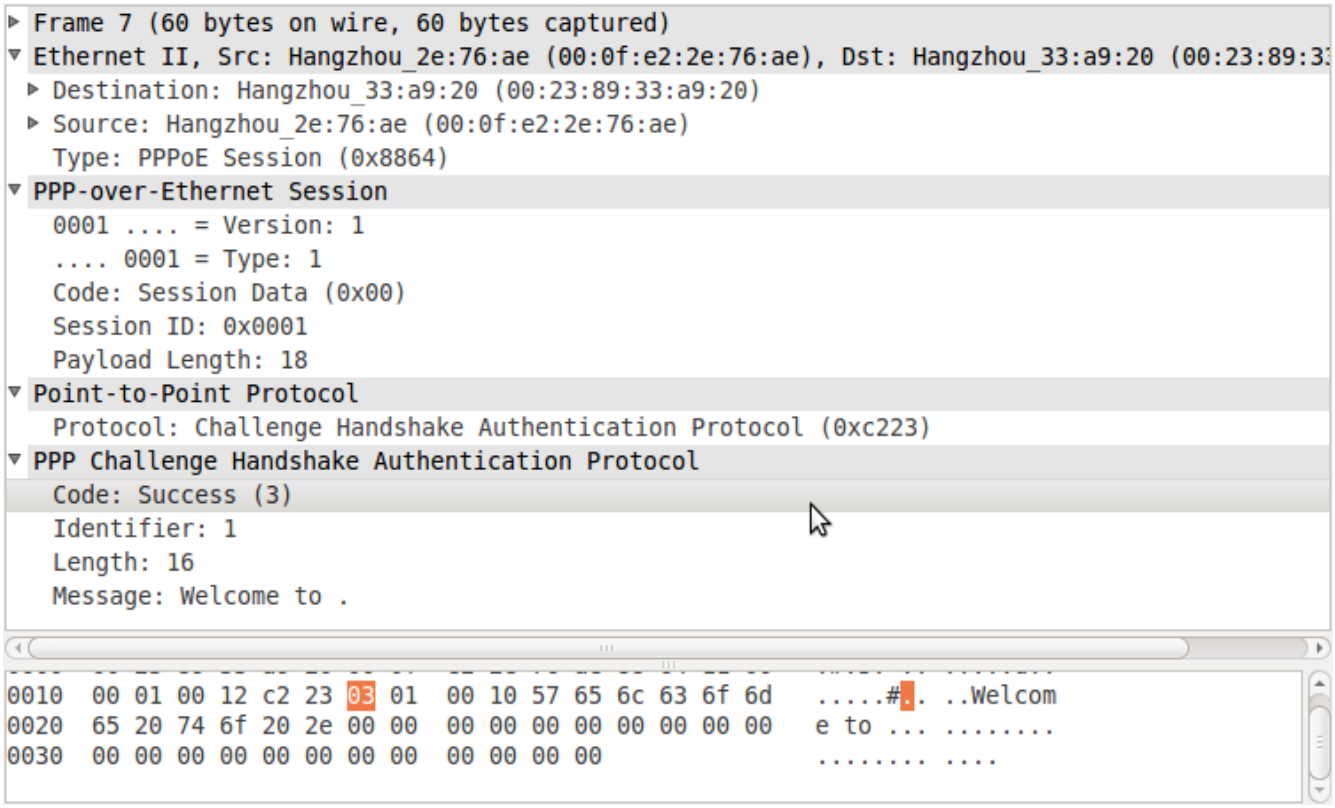


图 2-36 CHAP 的 *Success* 成功消息（认证方服务器发送）

- 成功消息的 Code 是 0x3，也携带与 Challenge、Response 相同的 Identifier 值，成功携带一句欢迎的话“Welcome to ”，这应该是厂家实现上的小 bug，本来是想写“Welcome to use this device”的，但长度定死了 16 字节，不够用，所以只剩下“Welcome to ”了，不过这并不影响最终结果，机器是看不懂这到底是欢迎还是不欢迎。

PPP 建立好 Session 后，需要通过一定的手段来维护 Session，这个手段是在 LCP 层实现的，也就是链路检测：

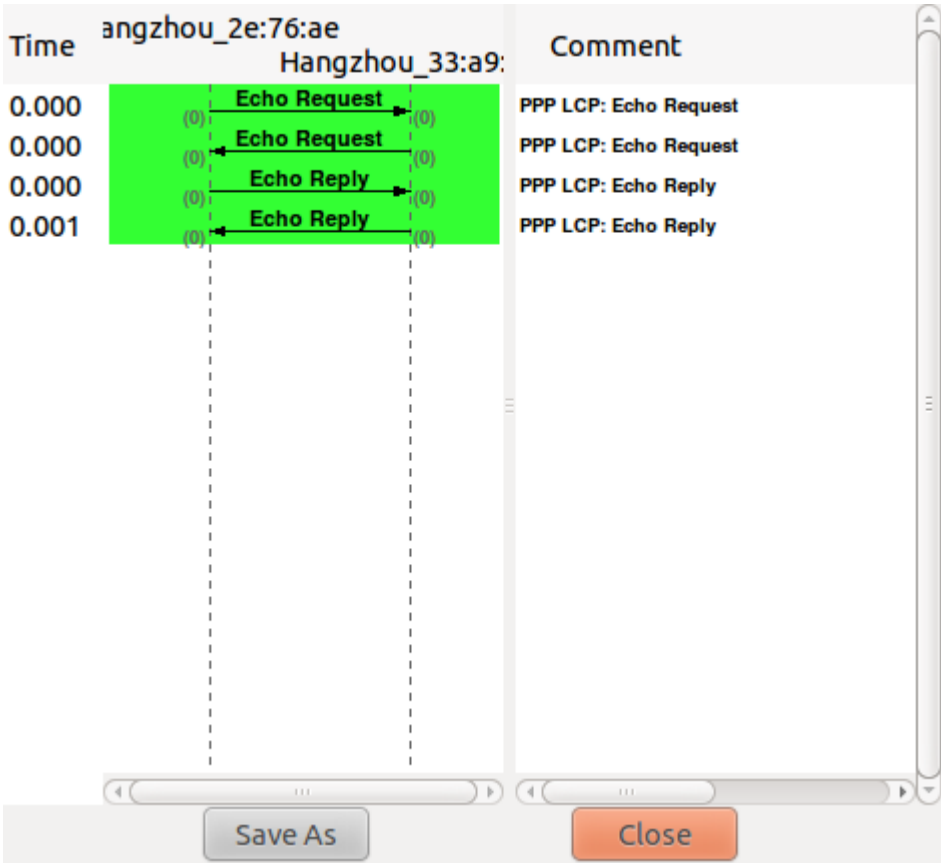


图 2-37 PPP LCP 回声检测系统（左为服务器，右为客户端）

Echo 回声是一个比较简单的物理现象，这个原理被运营在雷达上，可以探测航空器，用于激光探测地月距离也是一种回声，根据习惯，网络上的探测因此也被命名为 Echo 回声，PPP 上 LCP 的探测叫 LCP Echo，分为 *Echo Request* 回声请求和 *Echo Reply* 回声回应，从图 2-37 来看探测是双向探测，即服务器发送请求、客户端回应以及客户端发送请求、服务器回应：

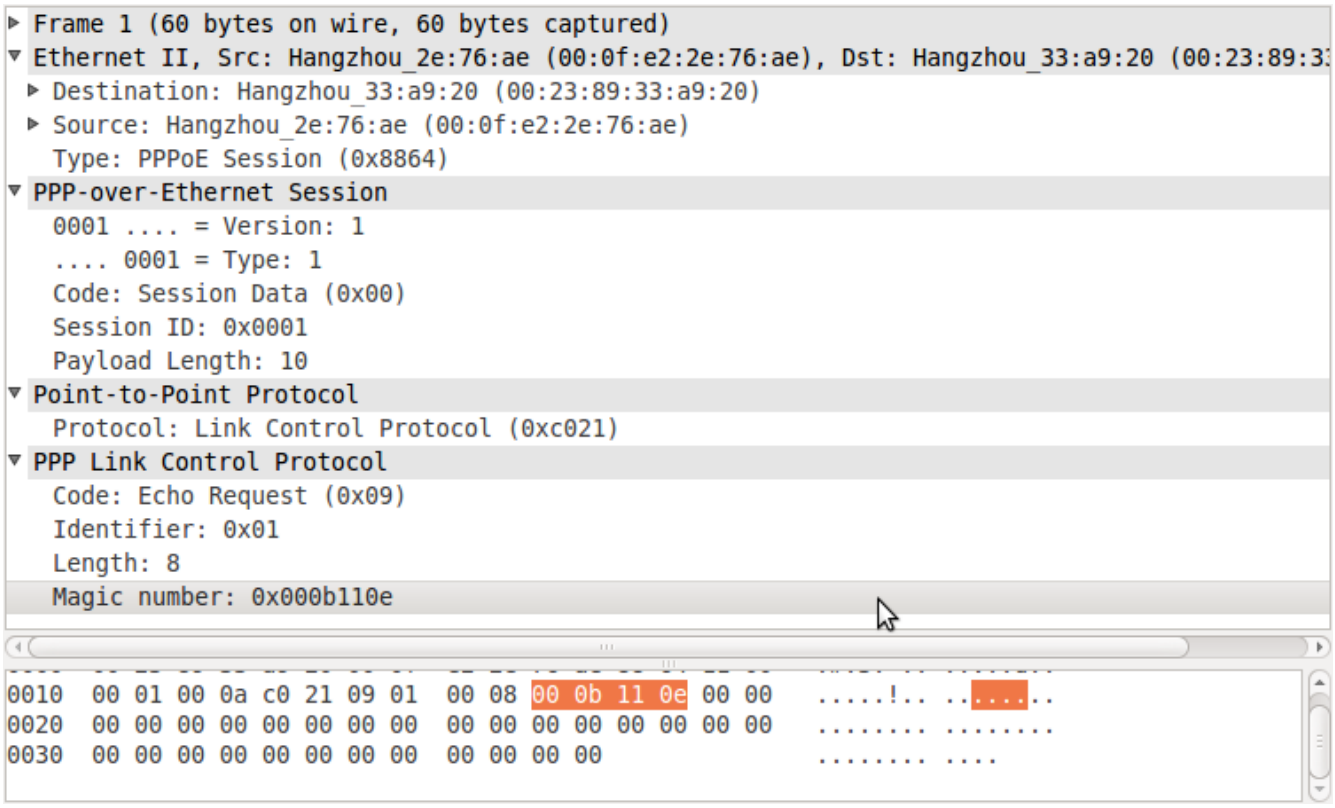


图 2-38 PPP LCP 回声请求（服务器发送）

- 关键信息是 LCP 中的 Code 是 0x09，表示 Echo Request 消息；
- Identifier 用来判断 Echo Reply 是否和 Echo Request 是否匹配；
- Magic Number 用来判断是否存在环路。

下面可以看一下客户端的 Echo Reply 是否和服务器的 Echo Request 相匹配：

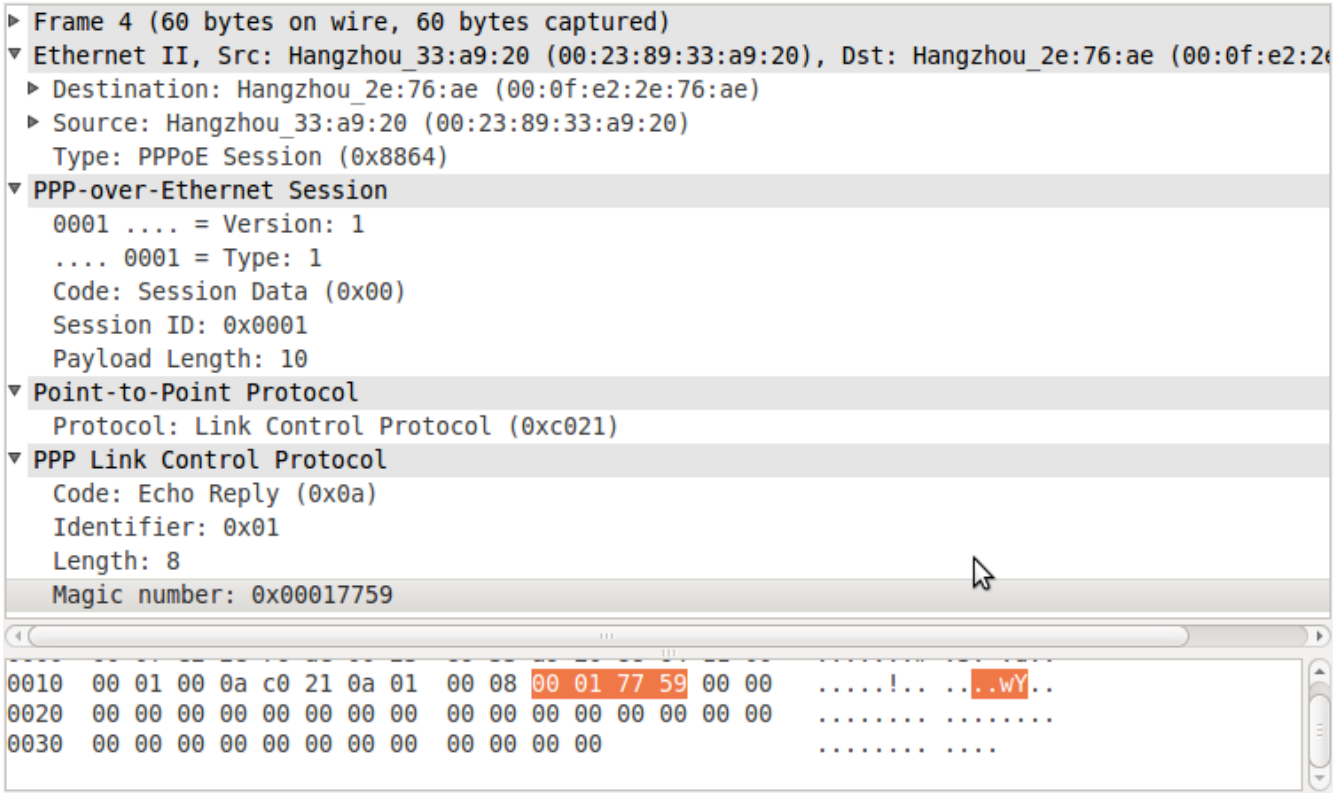


图 2-39 PPP LCP 回声回应（客户端发送）

- Echo Reply 对应的 Code 是 0x0a；
- Identifier 和服务器发送的 Echo Request 一致，是相匹配的。

至此 PPP 的大部分过程都介绍完成了。那么 PPP 会话是不是也像人一样，也会有挂的一天，PPP 的挂断是由 PPP LCP 完成的，叫 *terminate* 终止，终止也是有请求的：

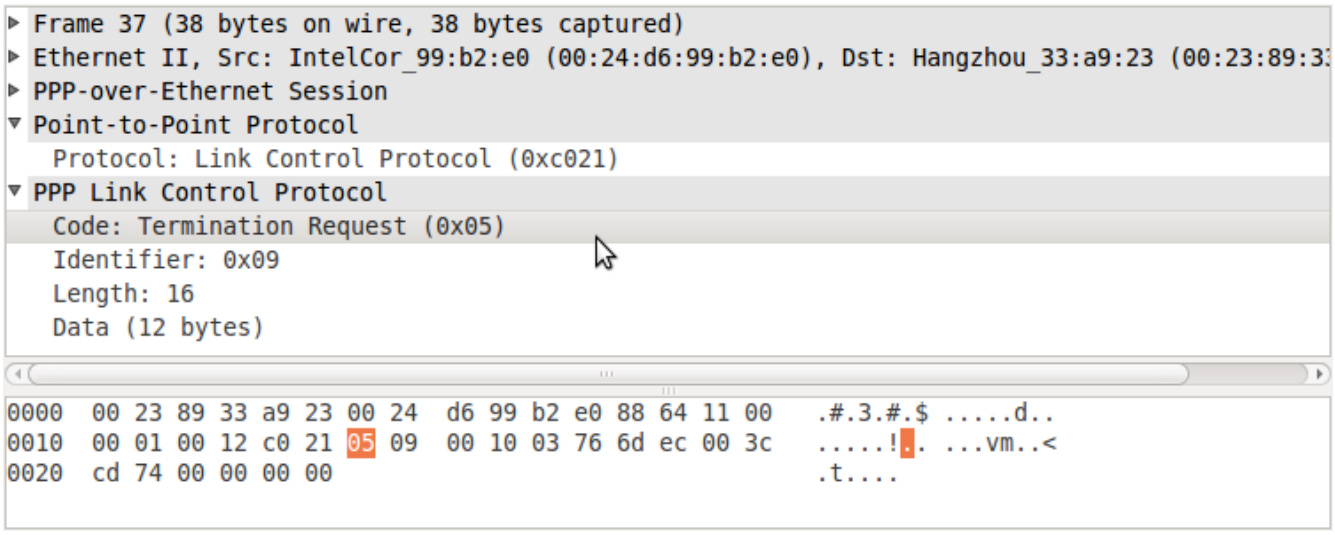


图 2-40 PPP LCP 终止请求（*客户端发送）

- LCP 中的 Code 是 0x05，表示 *terminate request* 终止请求；
- Identifier 用于确认终止应答；

- Data 字段是一些随机数，和不同的厂家实现有关。

服务器收到客户端的终止请求后通常会给予 *terminate ack* 终止确认：

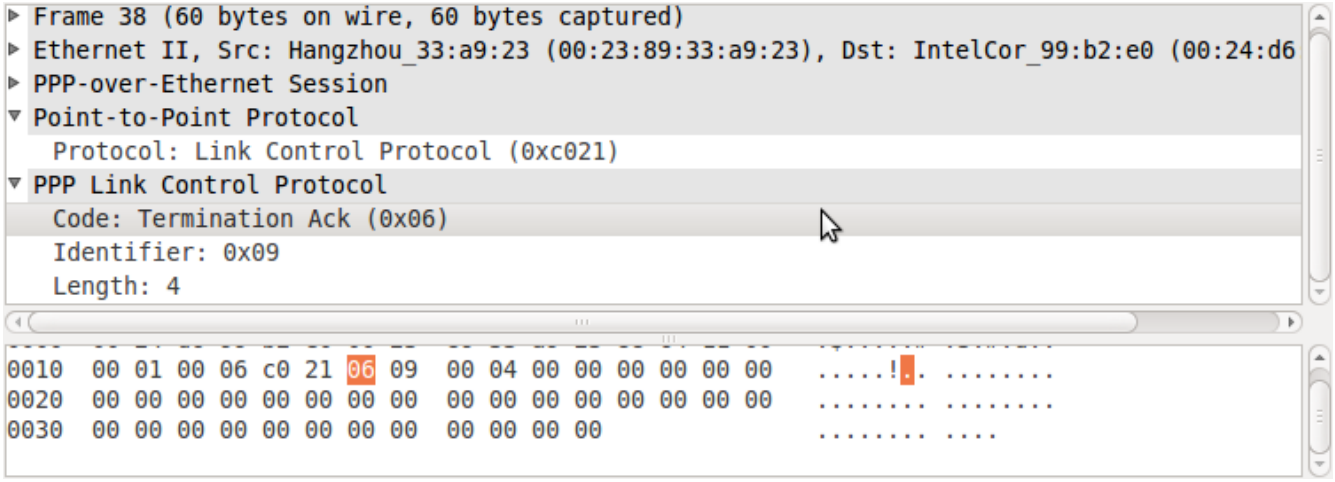


图 2-41 PPP LCP 终止确认（*服务器发送）

- LCP 的 Code 是 0x06，表示 *terminate ack* 终止确认；
- Identifier 和终止请求保持一致；
- 没有 Data 字段，因为这不是必须的。

通常 PPP LCP 的终止过程并不是必须的，因为很多时候 PPP 因为人为设备断电、拔线或者线路故障导致 PPP 会话终止，客户端来甚至来不及发送 *terminate request*，这个时候服务器 LCP 终止是靠连续 LCP Echo 探测失败（如服务器连续 3 个 LCP Echo Request 都失败了）来完成的，实验网络就是这种情况，图 2-40 和图 2-41 是另外单独抓取的正常终止。专家们在设计协议的时候都要考虑这种石沉大海似的异常故障处理机制。在很多厂家实现中，只要 PAP、CHAP 或者 NCP 阶段协商失败都要求从 LCP 开始重头协商，而不只是 LCP 协商失败。

PPP 终止了，是不是 PPPoE 也应该有所表示，没错，这就是 *discovery terminate* 发现终止过程，这个过程和 PPP LCP terminate 过程一样并不是经常可以看到，通常的的实现是 PPP LCP 终止，PPPoE 会话也就终止了。下面就来看一下 PPPoE 正常的终止（客户端主动发送给服务器）过程：

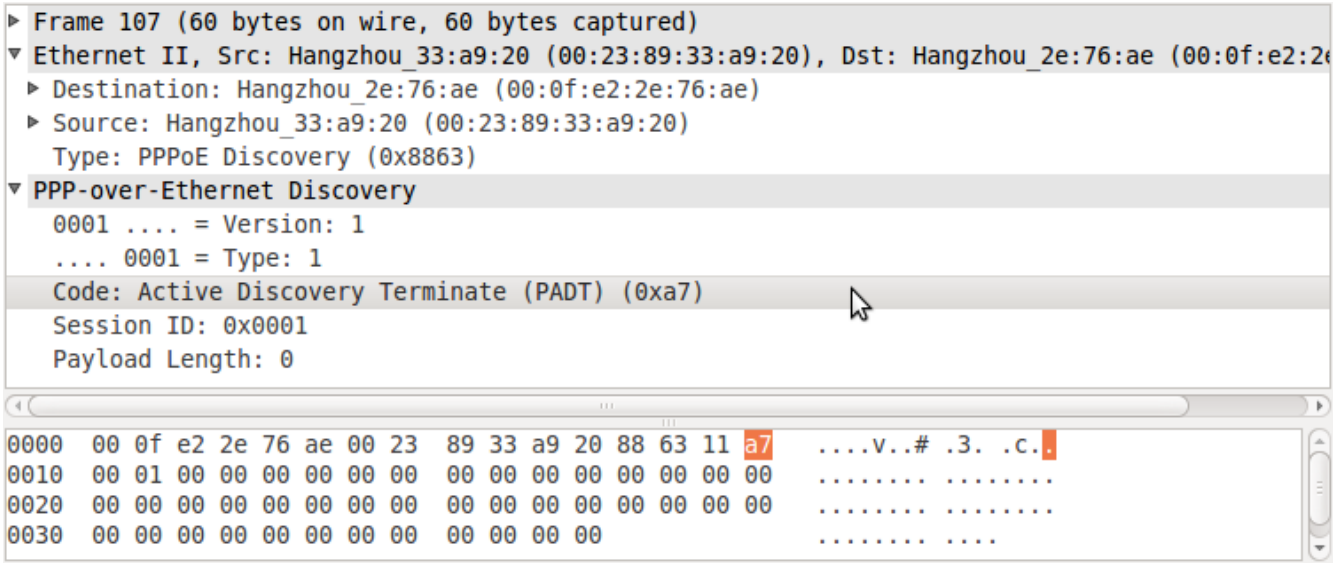


图 2-42 PPPoE *Discovery Terminate* 发现终止（客户端发送）

- 终止消息正确的说法是发现终止，以太网 *Type* 也是 0x8863；
- PPPoE 中的 Code 是 0xa7，表示这是发现终止消息；
- Session ID 告诉服务器客户端终止的是哪个会话。

这个消息只发送 1 次，服务器通常也会发一个同样的消息给予应答，其实客户端通常会等待一段时间，如果服务器实在没有什么表现，那客户端也不会傻傻地等着，将会话相关表项给予删除。

2.11. 802.1Q Virtual LAN（虚拟局域网）

以太网和 WLAN 都是 LAN 协议，里面涉及到链路 link，通常一条 link 就是一个 LAN，不同的 link 就是不同的 LAN，而网桥 bridge 可以將若干条 link 连接起来形成 1 条 link，也就是同一个 LAN，在网络的组建过程中通常是按照一定的使用需求划分 LAN，如相同行政单位使用相同的 LAN，不同的 LAN 之间通过 router 相连。既然有链路相连的需求，也会有链路分离的需求，特别是自从以太网交换机（也是一种 bridge）面世后，介质和链路相互脱离，一条链路可以包含若干个介质（物理端口），以太网交换机通常拥有较多的物理端口，因此可以將不同的端口划入到不同的链路中去，这种使用相同网络设备，但却属于不同链路的方式就叫 *Virtual LAN* 简称 VLAN，VLAN 在 802.1Q 中定义。先看一下 802.1Q 的封装格式（以以太网封装格式为例）：

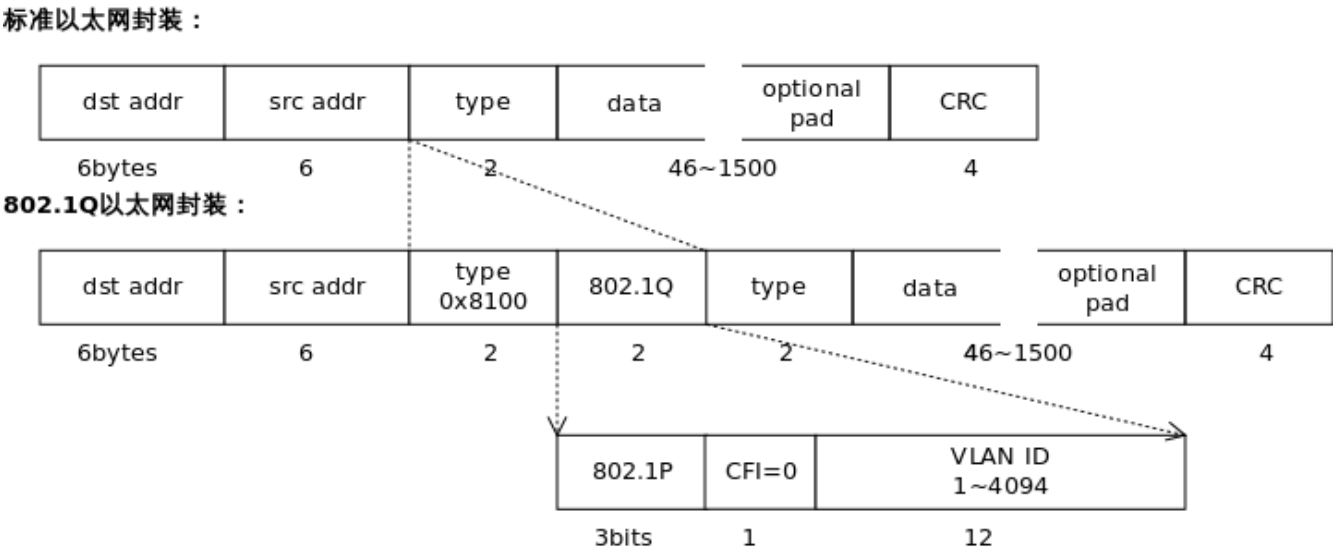


图 2-43 标准以太网封装和 802.1Q 以太网封装的区别

从图 2-43 可以发现 802.1Q 以太网封装（通常也被称为带 VLAN tag 的封装）有 2 个 type 字段：

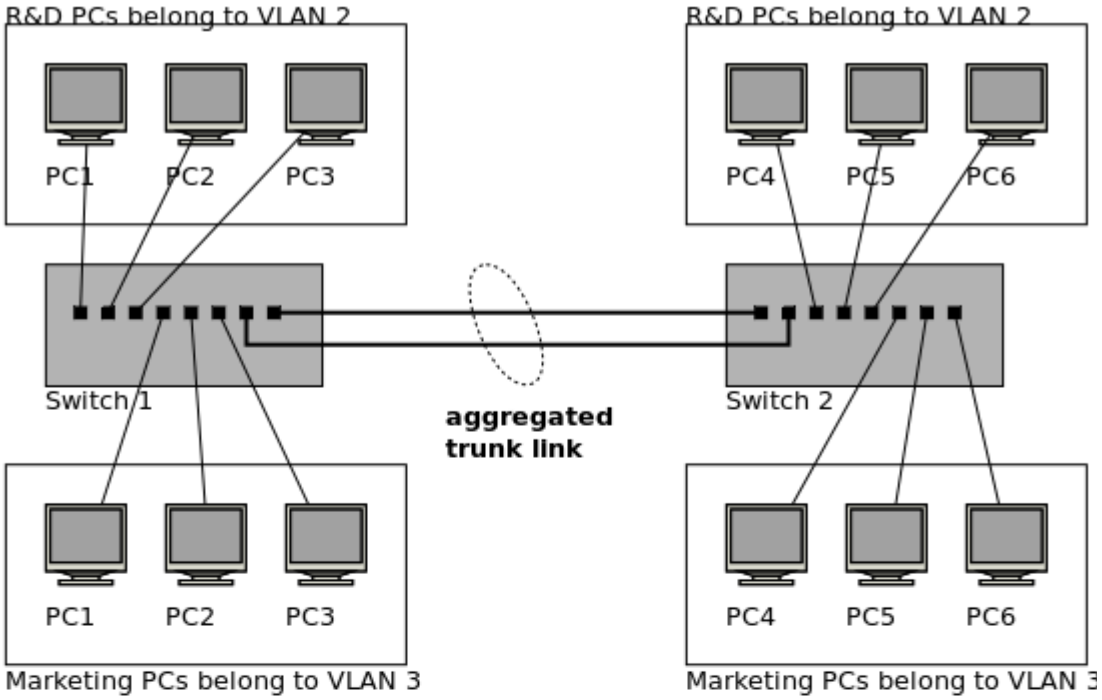
- 第 1 个 type 字段为 0x8100，用于指名这是一个 802.1Q 封装的以太网帧；
- 第 2 个 type 字段用于标识 data 类型，即标准以太网封装的 type 字段；

可见 802.1Q 封装以太网比标准封装多了 4 个字节，即 2 字节的 type（固定值 0x8100）和 2 字节的 802.1Q。802.1Q 由 3 个子字段构成：

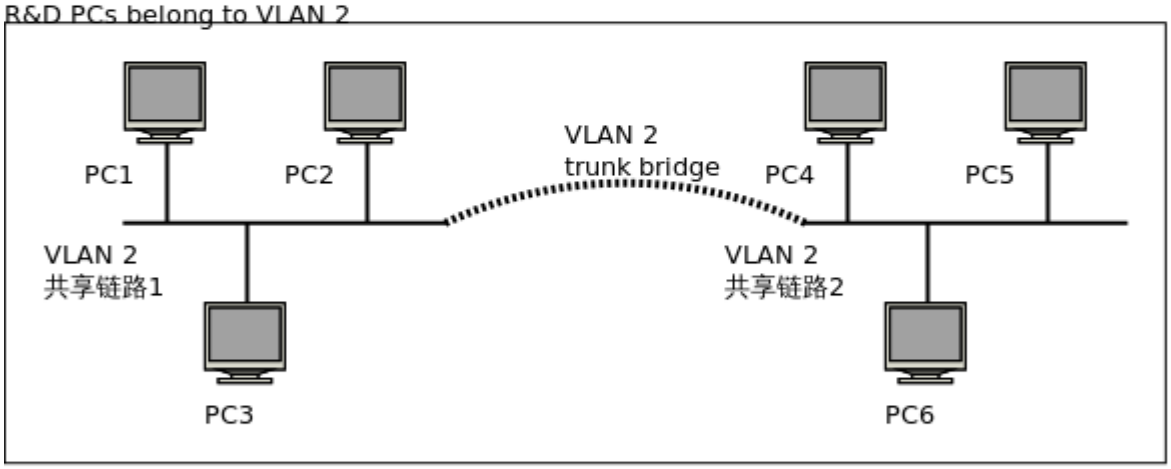
1. 3 位长的 802.1P，用于 QoS 中体现出优先等级，3 位长可以定义 0～7 共 8 个优先等级，默认值为 0；
2. 1 位长的 CFI，此位设 0；
3. 12 位长的 VLAN ID，共可定义 VLAN Id 范围 0～4095，其中 0 和 4095 保留，不可使用，能使用的为 1～4094，VLAN 1 通常为所有设备默认 VLAN。

VLAN 通常为交换机内部使用或者不同交换机之间使用，也就是网桥使用的，如下图所示：

802.1Q物理拓扑：



802.1Q逻辑拓扑：



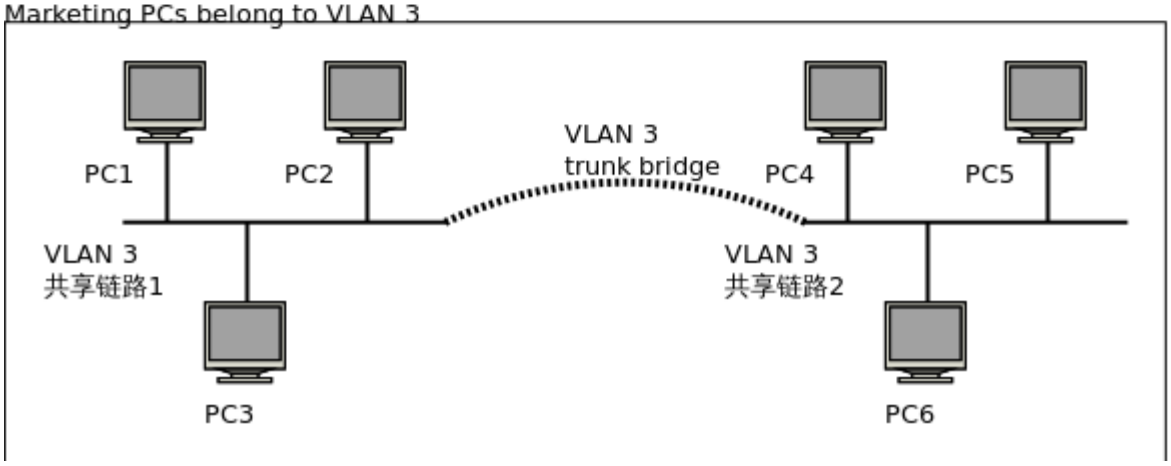


图 2-44 使用 VLAN 的组网拓扑模型

某个单位有两个部门 R&D 和 Marketing，各自都有组建 LAN 需求，该单位每楼层设置一台交换机，R&D 和 Marketing 也分布在不同楼层，此时可以给 R&D 的 PC 设置在 VLAN 2，Marketing 的 PC 则设置在 VLAN 3，不同楼层之间交换机通过 *trunk*（干线，如图 2-44 物理拓扑所示，通常设计网络时设备互联的 trunk 链路使用多条线路通过 *aggregated* 聚合方式互联）互联：

- PC 所连的端口是携带 VLAN 属性的，即从该端口进入的数据帧在交换机内部以及 trunk 线路上都是携带 VLAN tag 的；
- 交换机根据每个数据帧的 VLAN tag 来确定该数据帧可以在哪个 VLAN 内传输，如 Switch 1 的 R&D PC 在内部都是携带 VLAN ID = 2，那么通过 trunk 传递到 Switch 2 后，Switch 2 根据 VLAN ID = 2 判断，该数据帧只能在 VLAN 2 内传播，也就是 R&D 的 PC；
- 从交换机端口发到 PC 的端口通常会剥离 802.1Q，也就是 PC 收发都是标准以太网封装（这种端口被称为 *access* 模式；
    - access 模式：该端口只能属于 1 个 VLAN n，从该端口进入交换机的帧都会打上 VLAN ID = n 的封装，从该端口离开交换机的帧都会将 VLAN ID = n 的封装拆除；
    - trunk 模式：该端口可以属于若干个 VLAN，列如 x/y/z，其中 1 个为 primary VLAN，该 VLAN Id 被简称为 pvid，假设这里为 x，那么只允许 VLAN ID = x/y/z 的数据帧从该端口通过，其中 VLAN ID = x 的数据帧在离开交换机端口时会被拆掉 802.1Q 封装，变成标准以太网封装，而无 802.1Q 封装的数据帧从该端口进入交换机则会被添加 802.1Q 封装，VLAN ID = x；
    - hybrid 模式：该段口可以属于若干个 VLAN，和 trunk 的 pvid 一样，hybrid 也有 pvid，和 trunk 模式不同的是，从 hybrid 端口离开交换机时，可以再指定若干个 VLAN，將其 802.1Q 封装拆除，这些 VLAN 被称为 untagged VLAN，从 hybrid 端口离开交换机时为被拆除 802.1Q 封装的 VLAN 被称为 tagged VLAN，而从 hybrid 端口进入交换机的数据帧若无 802.1Q 封装则通通被打上 pvid 的 802.1Q 封装，列如某 hybrid 端口 pvid = x，untagged VLAN = y，tagged VLAN = z，那么 802.1Q 封装中 VLAN ID = x 和 y 的数据帧离开 hybrid 端口时都变成了标准以太网封装，只有 VLAN ID = z 的数据帧是携带 802.1Q 封装离开的；而从该端口进入的标准以太网帧都封装 802.1Q，其 VLAN ID = x，从该端口进入的 802.1Q 以太网帧，其 VLAN ID 只能是 y 和 z，否则会被丢弃；
- 交换机的端口模式是可以设定的，因此细致规划端口模式及所属 VLAN 是网络规划中很有学问的一项内容。

如逻辑拓扑所示，交换机内不同的 VLAN 之间因为被逻辑上相互隔离是无法相互通信的，因为不同的 VLAN 就是不同的链路（通常不同 VLAN 被分配以不同的网络信息），跨链路通信只能通过 router 或者 bridge 帮忙，交换机本身就是一种 bridge，交换机设置不同 VLAN 的目的就是划分出不同链路，如果此时

再用 bridge 将两个 VLAN 相连，那么直接在交换机上取消 VLAN 设置岂不是更方便，所以不同 VLAN 之间通信通常都是通过 router 实现的，现在许多交换机都具备了 router 功能（如果设备转发需要检查网络层信息，那么就是 router，如果只检查链路层信息，就是 bridge），这种交换机就被称为 *layer 3 switch*（俗称 3 层交换机）。目前大部分 PC 的以太网适配器（俗称网卡）都是不识别 802.1Q 的，可以通过适配器厂家提供的方法设置后可以支持，但不同厂家设置方法并不相同。目前许多服务器主机使用的以太网适配器慢慢开始流行打开 802.1Q 功能了，以提供更灵活的解决方案。值得一提的是 WLAN 的 AP 也通常是一种 802.1Q bridge（这是一种更流行的解决方案），从 STA 到 DS 的数据帧往往也被 AP 打上了 802.1Q 封装，从 DS 到 STA 的数据帧的 802.1Q 封装则会被 AP 拆除。下面就来看一下一个被 802.1Q 封装的 ARP request 数据帧的真身：

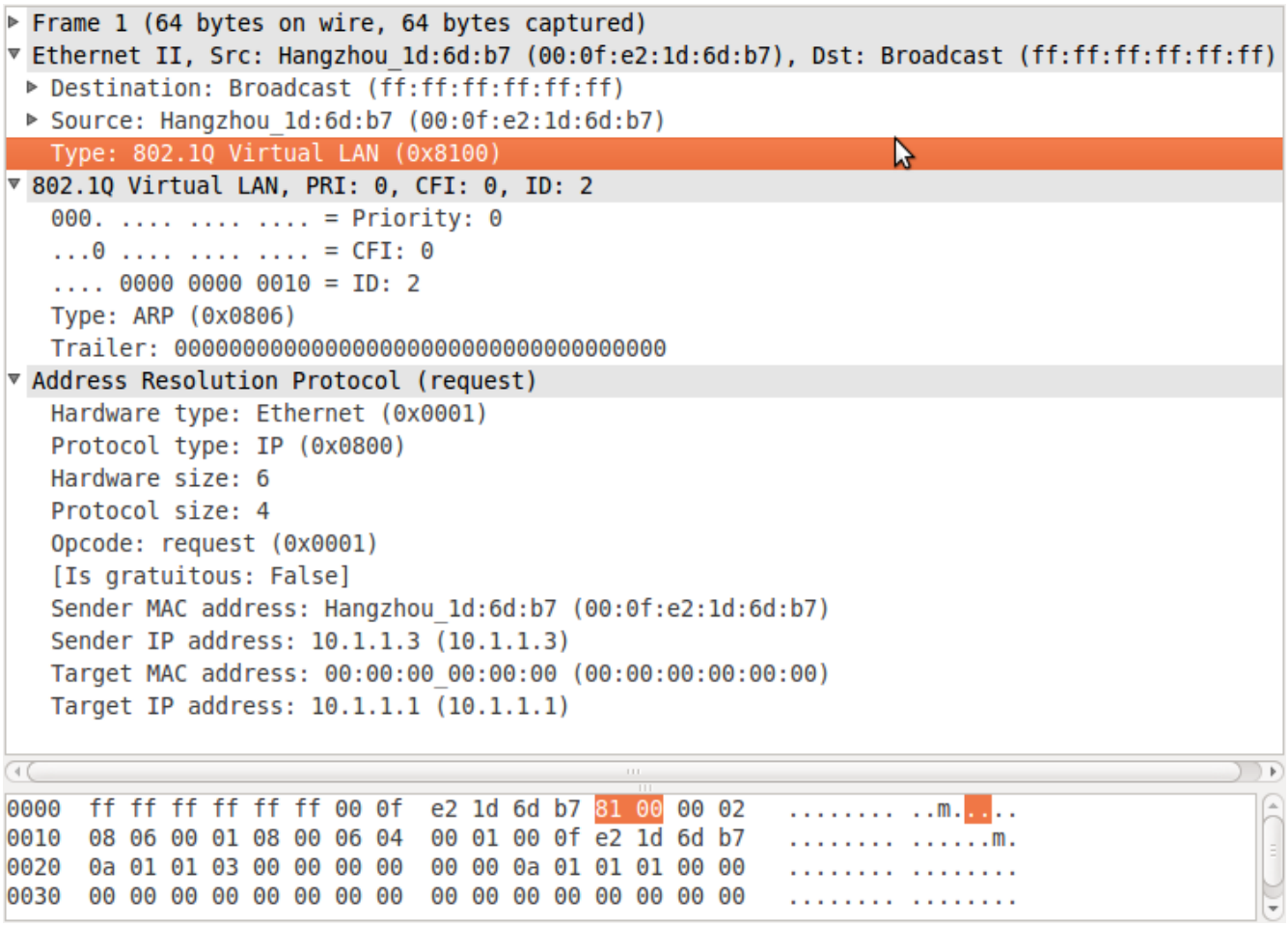


图 2-45 使用 802.1Q 封装的 ARP request

- 该以太网数据帧的第 1 个 type 可以看到是 0x8100，表明这是一个 802.1Q Virtual LAN 封装；
- 优先级 *priority* 是 0；
- VLAN ID = 2，说明这是个在 VLAN 2 内传播的数据帧；
- 第 2 个 type 字段值是 0x0806，表明 data 内容是 ARP 协议。

使用 802.1Q 封装后虽然多了 4 字节封装开销，但并没有缩减 data 字段的范围，也就是说 data 字段依然是

46～1500 字节。更有趣的是 802.1Q 封装还可以相互嵌套，即 2 层 802.1Q 封装（3 个 type 字段，2 个 802.1Q 封装），这种相互嵌套的 802.1Q 封装被称为 QinQ，意思是 802.1Q in 802.1Q。

2.12. 802.1X Authentication（802.1X 认证）

在 TCP/IP 诞生之前，计算机通信普遍采用专用网络专用协议，TCP/IP 以标准化的方式打破了计算机互联的限制，吸引了越来越多单位和个人把自己的计算机通过 IP 互联起来，而开放接入型以太网的诞生则使构建物理互联网络变得更加便捷，也更加便宜，可以说 TCP/IP 和以太网是一荣俱荣的关系。随着使用网络的人越来越多，就开始考虑网络性能和安全问题了（人都具备一种心理，如果拥有一个独特的玩意，会容忍它的各种缺点，如果这个玩意一旦开始流行，就会开始挑刺），在 2.10 节介绍 PPPoE 时略微介绍了一下，以太网需要通过认证才允许接入的需求，但 PPPoE 给人的感觉还是有点别别扭扭的，以太网需要一种原生的基于端口的认证接入控制（*port-based network access control*），这个怎么理解呢？以太网发展到现在主要是以 PC 连接到交换机端口的方式，基于端口就是指 PC 只有通过一种认证后，才允许使用交换机端口连接入网络。IEEE 为这种需求定义了一种框架，就是 *802.1X Authentication*，日常说 1X 认证就是这个框架，802.1X 则基于 IETF 所制定 *EAP*（*Extensible Authentication Protocol* 扩展认证协议），下面是 802.1X 和 EAP 的关系：

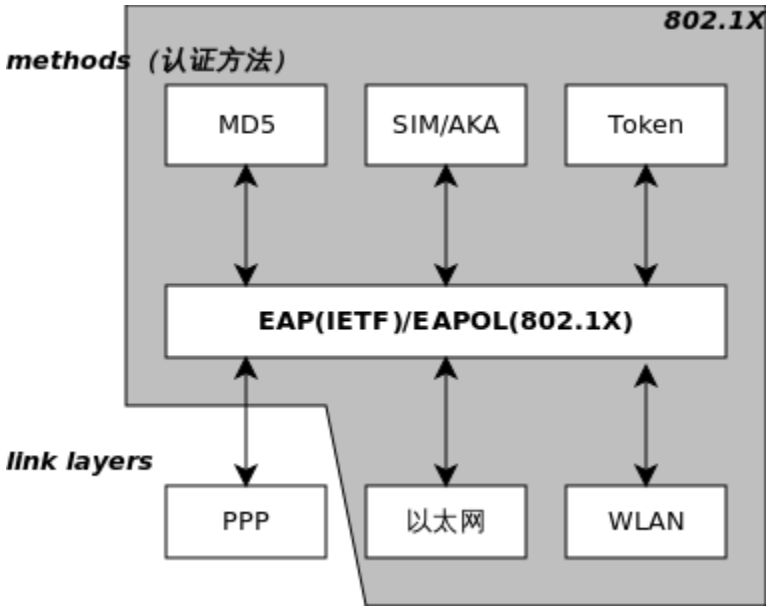


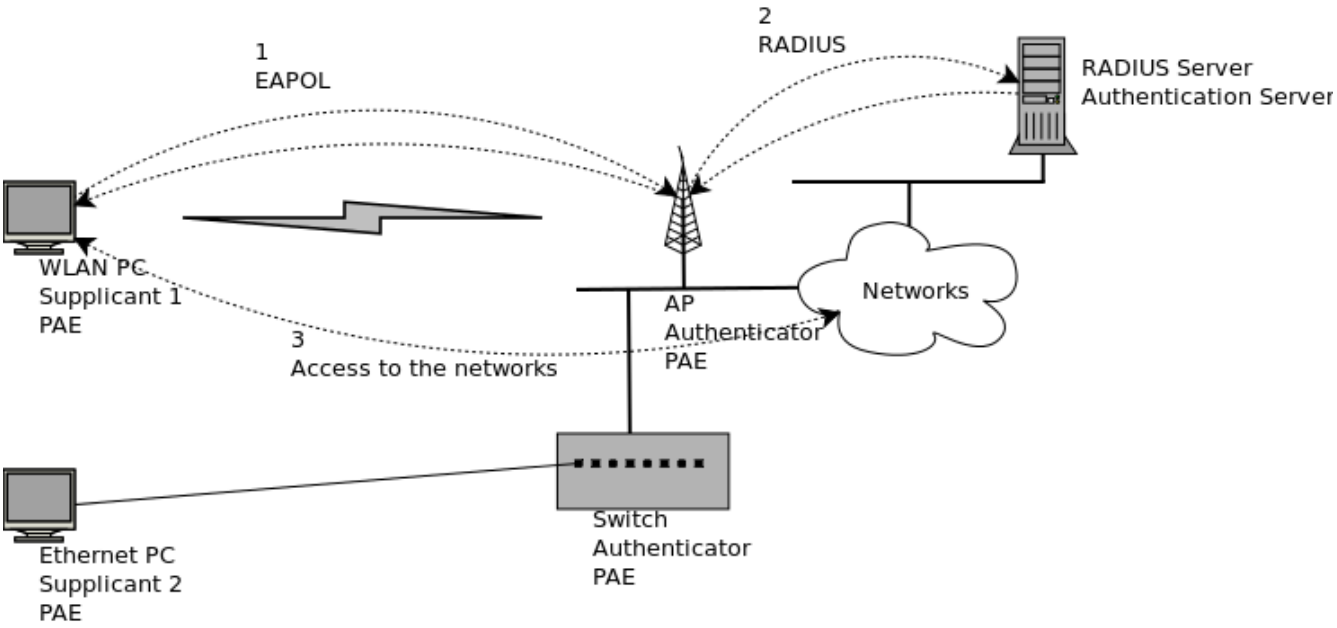
图 2-46 EAP 与 802.1X 关系

从上图看，EAP 是 IETF 定义的框架协议，目的是要建立一个独立于各种链路层的统一身份认证框架，如 PPP 中就使用的 PAP 和 CHAP，EAP 可以运行在 PPP 链路、以太网链路和 WLAN 链路上，EAP 并不需要任何网络层连接，恰恰是它运行的结果决定了网络层连接的成败，并且 EAP 并没有指定要使用什么样的认证方法，可以使用：

- 非加密验证方式，如 MD5，这种方式简单易行，但安全性不高，容易造成身份泄漏；
- 加密验证方式，如 *PKI*（*Public Key Infrastructure* 公共密钥基础设施）数字证书，被认为是非常坚固的安全架构，但加密验证方式也有许多种，如 *TLS*（*Transport Layer Security* 传输层安全）要求所有用户都拥有单独的数字证书，安全性极高，但部署起来也是极度复杂，*TTLS*（*Tunnel TLS*）和 *PEAP*（*Protected EAP*）则采取折中的方案，只需要拥有服务器证书即可建立 TLS 加密隧道，在 TLS 加密隧道内使用非加密验证方法，这种看似别扭的解决方案倒是迎合了许多用户的口味。

这也正好体现了 EAP 是一个框架性协议，给开发者、用户都留有非常灵活的选择空间，这看起来是个好事，其实未必，EAP 就是因为太活了，导致可供用户选择的解决方案太多，而且许多厂家的实现方式无法互相兼容，任何企业部署 EAP 的时候都面临许多设备选型和兼容性困难。而 802.1X 则是主要针对以太网和 WLAN 的 EAP，称为 *EAPOL*（*EAP Over LAN*），所以，802.1X 具备 EAP 同样的特点，非常灵活，没有绝对流行的解决方案，以至于国内更青睐于另外一种认证接入方案——基于 WEB 的 *Portal* 方案（不需要在每台接入设备上部署，只需要在 PC 的网关上部署，所以不涉及接入设备兼容性问题，但每个厂家的方案基本上都是私有的，无法互通，因此海外用户更青睐更标准一些的 802.1X）。802.1X 是针对端口的认证框架，也就是说认证前，以太网的端口只允许进行 802.1X 认证操作，其余业务数据全部禁止通行，认证通过后方能通行，而如 2.44 所示，交换机端口是可以再连接交换机的，假设交换机 1 所有端口开启了 802.1X 认证，此时用户在某个端口私自连接了交换机 2（交换机 1 并不能感知某个端口连接的是交换机还是 PC），然后在交换机 2 上连接若干 PC，那么只要交换机 2 上某个 PC 认证通过，交换机 2 上其余所有 PC 都能够访问网络（虽然网速有可能下降，但这种类似捡便宜的蹭网行为一直在民间有很大的吸引力）；此外 WLAN 使用的是共享介质通信，完全没有以太网交换机端口的概念；基于端口的访问控制并不是网络管理员最终需要的结果，他更需要的是一种基于 MAC 地址的认证接入控制（*MAC-based network access control*），即每台 PC 不管是以太网还是 WLAN，它的 MAC 地址都是不同的，针对每个 MAC 做认证可以控制的粒度就不再是某个端口，而是每台 PC，而目前市场上流行的 802.1X 都已经实现了基于 MAC 的接入控制（这可以看作是市场优胜劣汰的结果），也基本上是默认的方式。

在 802.1X 中，可以结合下图了解一下组成部分：



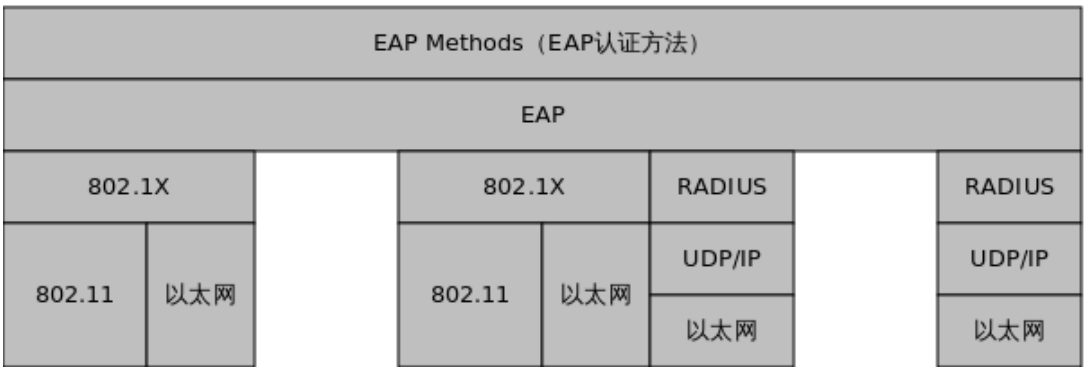


图 2-47 802.1X 体系简介

- *Supplicant*：802.1X 中的被认证端，从字面上理解就是客户端请求认证，以访问网络，支持若干种 EAP 方法，图 2-47 中，WLAN Pc 和 Ethernet PC 都是请求认证接入的 Supplicant；
- *Authenticator*：802.1X 中的认证端，也就是连接客户端的网络设备，位于网络的边缘，位于 Supplicant 和 Authentication Server 中间，是网络控制的最佳位置，与 Supplicant 一起运行 802.1X，和 Supplicant 一起并成为 802.1X 中的 *PAE*（*Port Authentication Entity* 端口认证实体），图 2-47 中，连接 WLAN PC 的 AP 和连接 Ethernet PC 的 Switch 都是 Authenticator；
- *Authentication Server*：802.1X 体系中认证体系的核心组件，保存有允许接入网络的所有身份及保密信息，支持 EAP 若干方法，有不少厂家可以做到 Authenticator 内置 Authentication Server，但仅支持一些简单的 EAP Method。

再来看一下信息封装：

- Supplicant 和 Authenticator 之间通常运行的是 802.11 或以太网，而 802.1X 就可以欢快地运行在这两种链路之上，802.1X 封装的是 EAP（就是 EAPOL），EAP 中则使用某种指定的 EAP Method；
- Authenticator 和 Authentication Server 之间通常是通过一个网络连接，因为一般而言，整个网络只会设立一套 Authentication Server，作集中式身份认证，Authenticator 將 Supplicant 封装在 EAP 中的秘密素材、认证信息封装到 *RADIUS*（*Remote Authentication Dial In User Service*，是一种基于 UDP 的身份认证、授权、计费协议，被广泛使用），同时 Authenticator 將 Authentication Server 返回的秘密素材、认证信息（以 EAP 方式封装在 RADIUS 中）通过 802.1X 再返回给 Supplicant；
- 从以上可以看出 802.1X 把认证工作主要交给 Supplicant 和 Authentication Server，Authenticator 只是个执行命令的看门者。

整个 802.1X 的完整过程是如下：

1. Supplicant 通过 802.1X 向 Authenticator 提交认证申请并提供一些身份信息；
2. Authenticator 通过 RADIUS 向 Authentication Server 提供 Supplicant 的身份信息；
3. Authentication Server 通过 RADIUS 向 Authenticator 返回认证结果；
4. Authenticator 检查认证结果，通过 802.1X 在向 Supplicant 通报结果的同时，执行结果相应的动作
    1) 如果认证通过，Authenticator 就对 Supplicant 开放非 EAPOL 帧的通过权限，Supplicant 终于可以干一些除了 802.1X 认证之外的事情——获得网络层（如 IPv4/IPv6）通信许可（网络层通信许可由 Authenticator 根据认证结果执行），也就是可以访问网络了；
    2) 如果认证不通过，那么 Supplicant 就比较悲剧，由于无法获得网络层通信许可，只能日复一日地和 Authenticator 进行枯燥的 802.1X 认证，访问网络只能是痴心妄想。

需要说明的是 802.1X 认证是对用户（Supplicant 操作者）进行认证，而不是对 Supplicant 机器本身，所以在认证过程中，需要一些互动，比如手工输入用户名、动态口令或者提前设置自动认证，又或者使用 Supplicant 操作系统账号自动认证等等，这么做的目的就是管理精细化。下面就以 MD5-Challenge 为 EAP Method 的 802.1X 交互过程（Supplicant 和 Authenticator 之间，RADIUS 交互本章不做介绍）：

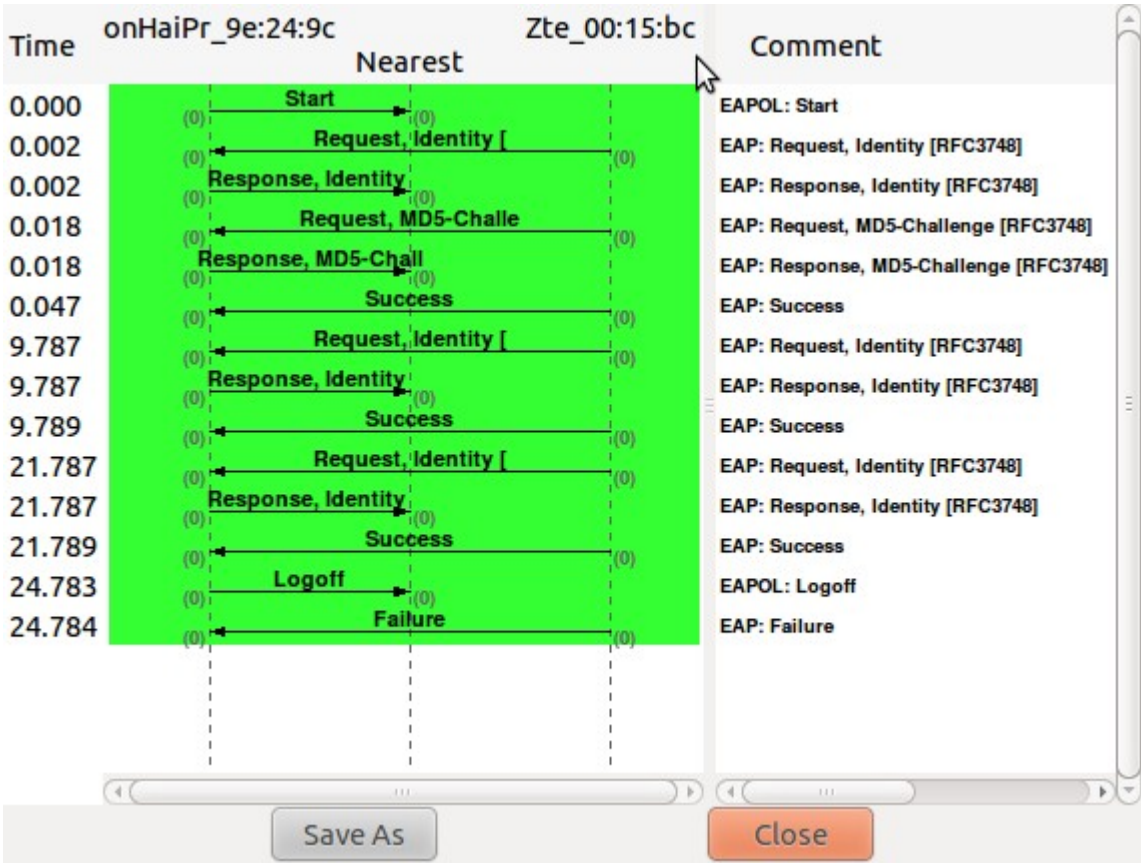


图 2-48 MD5-Challenge 方式 802.1X（左为*Supplicant，右为*Authenticator）

本抓包是从同事那要的，并非从实验网络中获取。Supplicant 发出的帧 DA 是一个熟知组播地址 *Nearest*（01:80:c2:00:00:03），这个地址代表所有 Authenticator，相反 Authenticator 直接回给 Supplicant，这是以太网的缘故，所有数据帧都只能从 Authenticator 的端口进出，所以使用熟知地址 Authenticator 依然可以收到，而在 WLAN 中，则使用 AP 的 MAC 地址作为 DA（也是 RA），只有 AP 是唯一的 Authenticator，为了避免其余 STA 也处理组播帧，所以使用单播方式：

```
▶ Frame 1 (64 bytes on wire, 64 bytes captured)
▼ Ethernet II, Src: HonHaiPr_9e:24:9c (00:1c:25:9e:24:9c), Dst: Nearest (01:80:c2:00:00:03)
  ▶ Destination: Nearest (01:80:c2:00:00:03)
  ▶ Source: HonHaiPr_9e:24:9c (00:1c:25:9e:24:9c)
    Type: 802.1X Authentication (0x888e)
    Trailer: 000000000000000000000000000000000000000000000000...
    Frame check sequence: 0x00000000 [incorrect, should be 0x82522534]
▼ 802.1X Authentication
    Version: 1
    Type: Start (1)
    Length: 0

0000  01 80 c2 00 00 03 00 1c  25 9e 24 9c 88 8e 01 01   ........ %.$.....
0010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```

图 2-49 EAPOL——Start

1. 如图 2-49，Supplicant 首先发出了一个 EAPOL 帧，类型叫 Start，意思再明确不过，就是开始申请认证了；

   1) 802.1X 在以太网和 WLAN 中的 Type 值是一样的，那就是 0x888E；

   2) 802.1X 中包含 1 字节 Version（目前有 1、2，1 更成熟），这里是 1；

   3) 802.1X 中也有一个 Type 字段，但只有 1 字节，这里是 1，表示 Start；

   4) 802.1X 中 2 字节长 Length 表示携带多少其余信息，此处为 0，表示未携带任何信息；

   5) 802.1X 部分总长 4 字节，远没有达到 Ethernet 46 字节最短 Payload 的要求，所以含有 42 字节的 PAD，在 Wireshark 中，PAD 被描述成 Trailer。

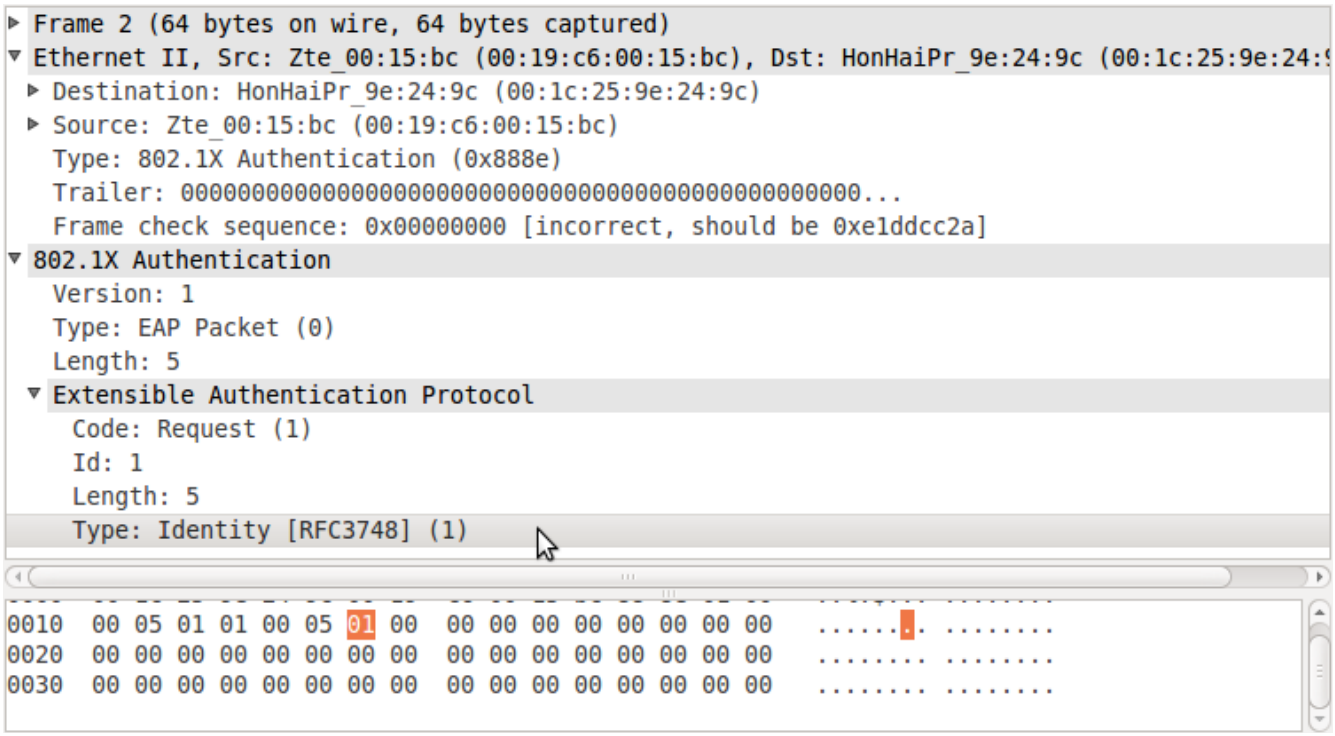


图 2-50 EAP——Identity Request（*Authenticator 发送）

2. 如图 2-50 所示，Authenticator 接收到 EAPOL 的 Start 后，立刻响应，请求 RFC 3748 格式身份：

   1) 802.1X 中 Type 是 0，表示 802.1X 中封装了 EAP；

   2) 802.1X 中 Length 值是 5，表示 802.1X 中封装的 EAP 信息长度为 5 字节；

   3) 看看 EAP 信息格式：

      (1) Code 字段，长度 1 字节，值为 1 表示 Request 请求；

(2) Identity 字段，长度 1 字节，用来匹配 Request 及 Response；

(3) Length 字段，长度 2 字节，表示 EAP 信息单元长度，这里值为 5，包括 Code、Identity、Length 和后面的 Data 字段，共 5 字节；

(4) Type 字段，长 1 字节，用于描述 Type-Data 的类型，该字段是否存在由 Code 决定，图 2-47 所述 EAP Methods 就在 Type 和 Type-Data 中携带，这里值为 1，表示使用 RFC 3748（该 RFC 定义 EAP）中定义的身份；

(5) Type-Data 字段，长度和内容取决于 Type 值，该字段长度可以从 Length 字段计算，Length 值 - Code 长度 - Identity 长度 - Type 长度就是 Type-Data 长度，这里很容易计算出来 Type-Data 字段是 0 字节长（Type 占 1 字节）。

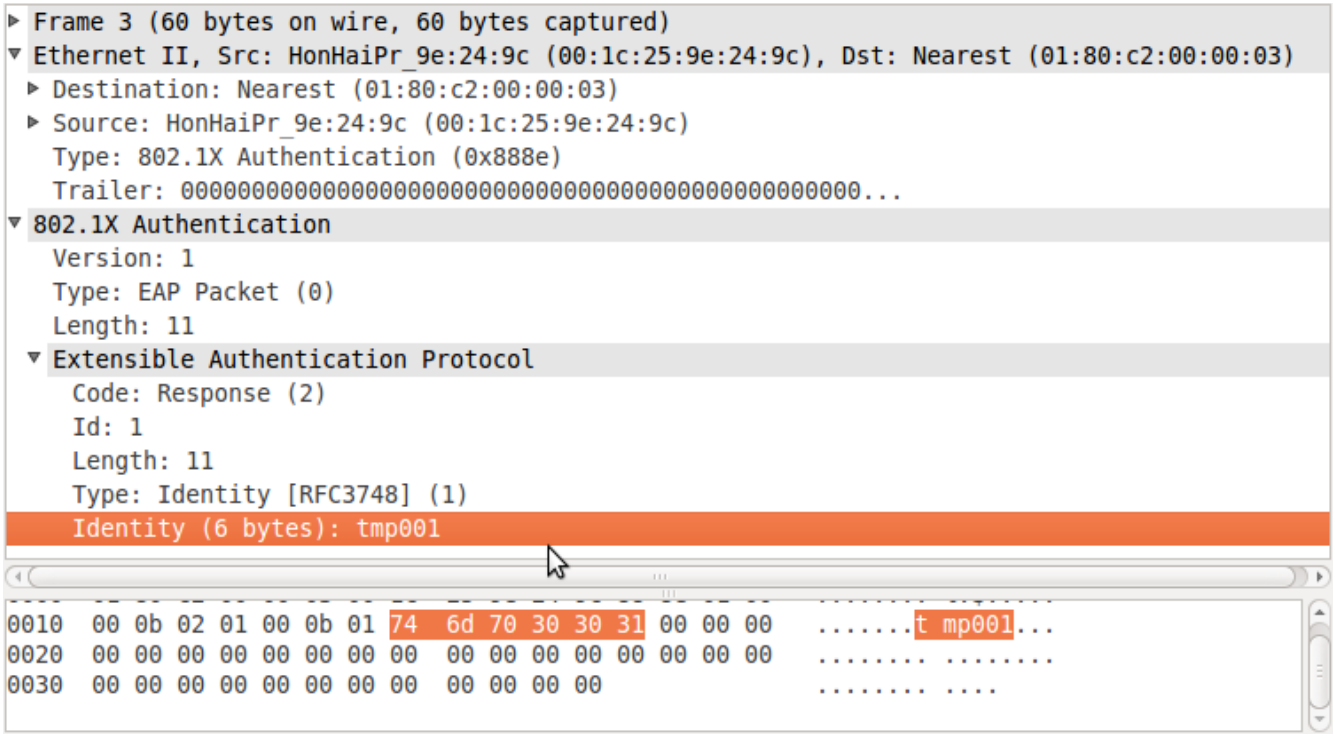


图 2-51 EAP——Identity Response（*Supplicant 发送）

3. 如图 2-51 所示，Supplicant 收到 Authenticator 的 Identity Request 后立刻以 Identity Response 响应：

1) 802.1X 部分和 Identity Request 相比，不一样的地方在于 Length，值为 11，这说明 EAP 部分有些不一样：

(1) Code 值为 2，表示 Response；

(2) Identity 部分和 Request 一致，都为 1，说明 Response 是对之前 Request 的回应；

(3) Length 部分也是 11；

(4) Type 和 Type-Data 部分，长度为 11 – 1 – 1 – 2 = 7（Bytes），其中 Type 部分值依然是 1，表示采用 RFC 3748 定义的 Identity，值为 tmp001，这个字符串长 6 Bytes，再加上 Type 长度 1 Byte，正好 7 Bytes，从这里我们可以发现 Supplicant 的身份是 tmp001。

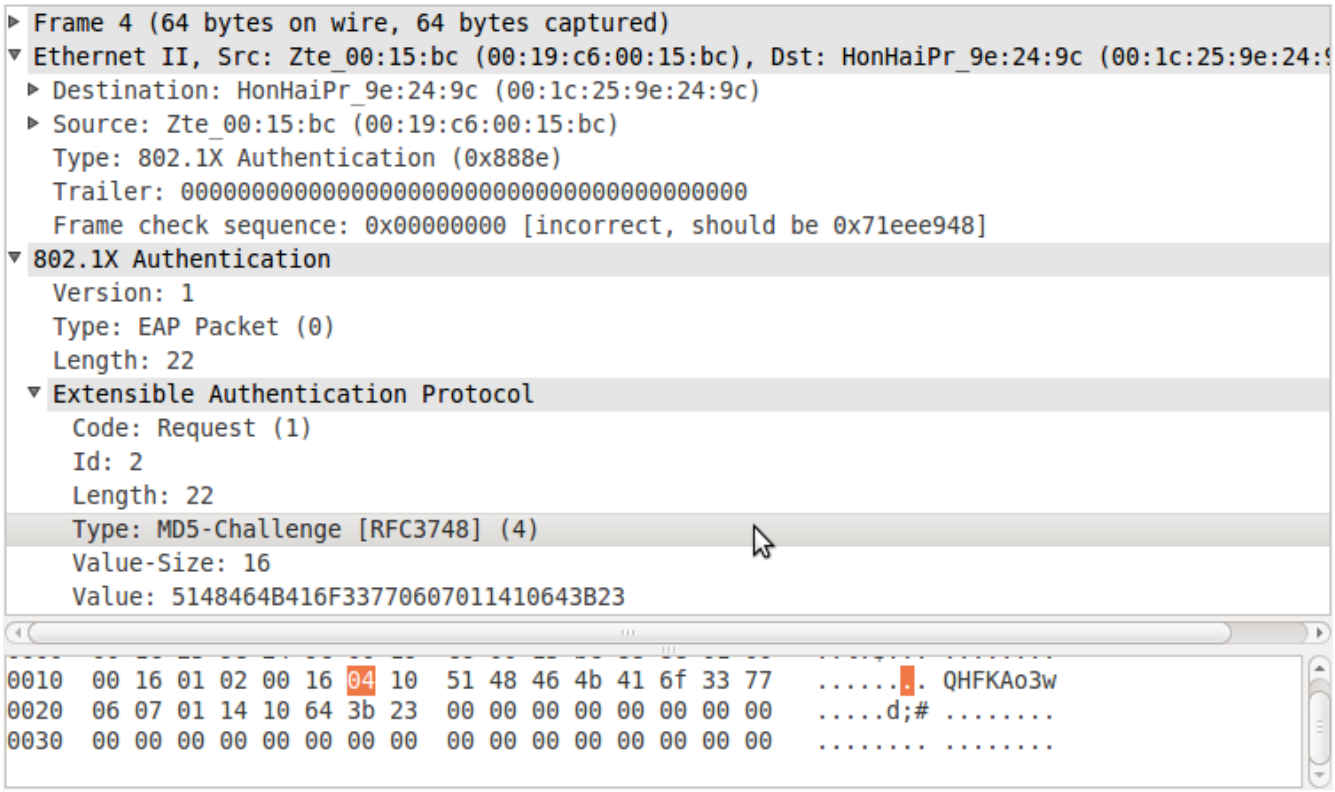


图 2-52 EAP——MD5-Challenge Request（*Authenticator 发送）

4. 如图 2-52 所示，当 Authenticator 得到 Supplicant 的身份信息（Identity）后，会将 Identity 发给 Authentication Server，Authentication Server 会根据身份类型和身份信息进行处理，即以 EAP 中的 MD5-Challenge Request（MD5 挑战请求）进行身份验证：

   1) Code 值回到了 1，表示 Request；

   2) Identity 此番变成了 2，该 Authenticator 的实现是按照每个 Request 递增 Identity；

   3) Length 变成了 22，Type-Data 部分长度为 18 Bytes：

Type	**EAP Method**（认证方法）	说明
4	MD5-Challenge	EAP 中类似于 CHAP 的认证方式（非加密）
6	GTC/Generic Token Card	一般的令牌卡，一种流行的动态同步口令（非加密）
18	EAP-SIM	使用 GSM 手机中的 SIM 卡作为认证身份信息（非加密）
23	EAP-AKA	3G 手机中使用的身份验证系统，*Authentication & Key Agreement*（认证与密钥协商），可以看作是 EAP-SIM 的升级

		版本
29	EAP-MSCHAP-V2	微软开发的一种域帐号认证 CHAP 方式（非加密）
13	EAP-TLS	Supplicant 和 Authentication Server 使用 PKI 的数字证书（digital certificate）相互认证（加密），如果再把 Authenticator 也拉进去进行 3 方相互认证，就和中国力推的 *WAPI*（*WLAN Authentication & Privacy Infrastructure*）方案组成部分之一相同了。
21	EAP-TTLS	Tunnel TLS，使用 TLS 隧道加密保护较弱的身份验证方式（加密），和 EAP-TLS 不同的是，TLS 使用数字证书进行双向认证，而 TTLS 则只有 Authentication Server 有数字证书，Supplicant 利用该证书与 Authentication Server 建立 TLS 隧道，在隧道内部使用非加密方法对 Supplicant 身份进行认证，所以 TTLS 目前要比 TLS 更容易部署。
25	PEAP	Protected EAP（加密），原理和 TTLS 类似，但不如 TTLS 灵活，PEAP 在 TLS 隧道中只能使用 EAP（RFC 3748）定义的方法。

(1) Type 值是 4，意思是 Type-Data 部分为 RFC 3748 中定义的 MD5-Challenge，这就是所说的 EAP Method，其余值如上表所示；

(2) Type-Data 长度是多少呢，专门由 1 Byte 长的 Value-Size 来描述，这里显示是 16 Bytes；

(3) Value 部分就是 16 Bytes 长 MD5-Challenge 的数值。

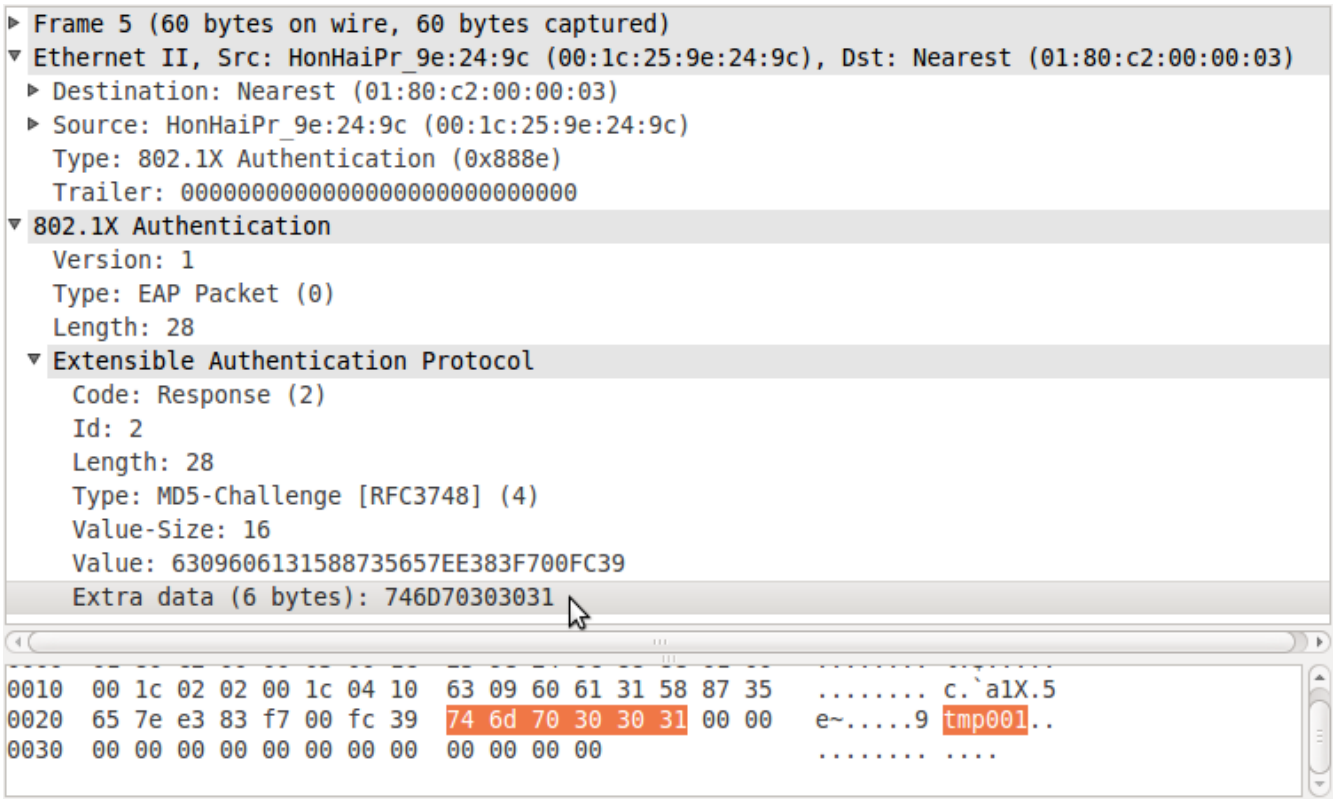


图 2-53 EAP——MD5-Challenge Response（*Supplicant 发送）

5. Supplicant 一旦收到 MD5-Challenge Request 就立刻要用 Response 进行响应，不然就不要想连接到网络了，EAP 部分结构如图 2-53 所示：

   1) Code 值为 2，表示 Response 类消息；

   2) Identity 也为 2，但是对 Request 中 Identity 的回应；

   3) Length 为 28，Type-Data 则由被由 4 部分构成：

      (1) Type 依然是 4，表示 RFC 3748 定义的 MD5-Challenge

      (2) Value-Size 依然是 16；

      (3) Value 是根据 MD5-Challenge Request 中 Value 和 Supplicant 身份信息、身份验证信息（俗称用户口令）、Identity 字段使用 MD5 计算的结果；

      (4) 最后多出了一个 6 字节的 Extra Data，值是 tmp001，这其实是 Supplicant 的身份信息，所以 Response 的 Length 值是 28，要比 Request 中 Length 多 6 的缘故。

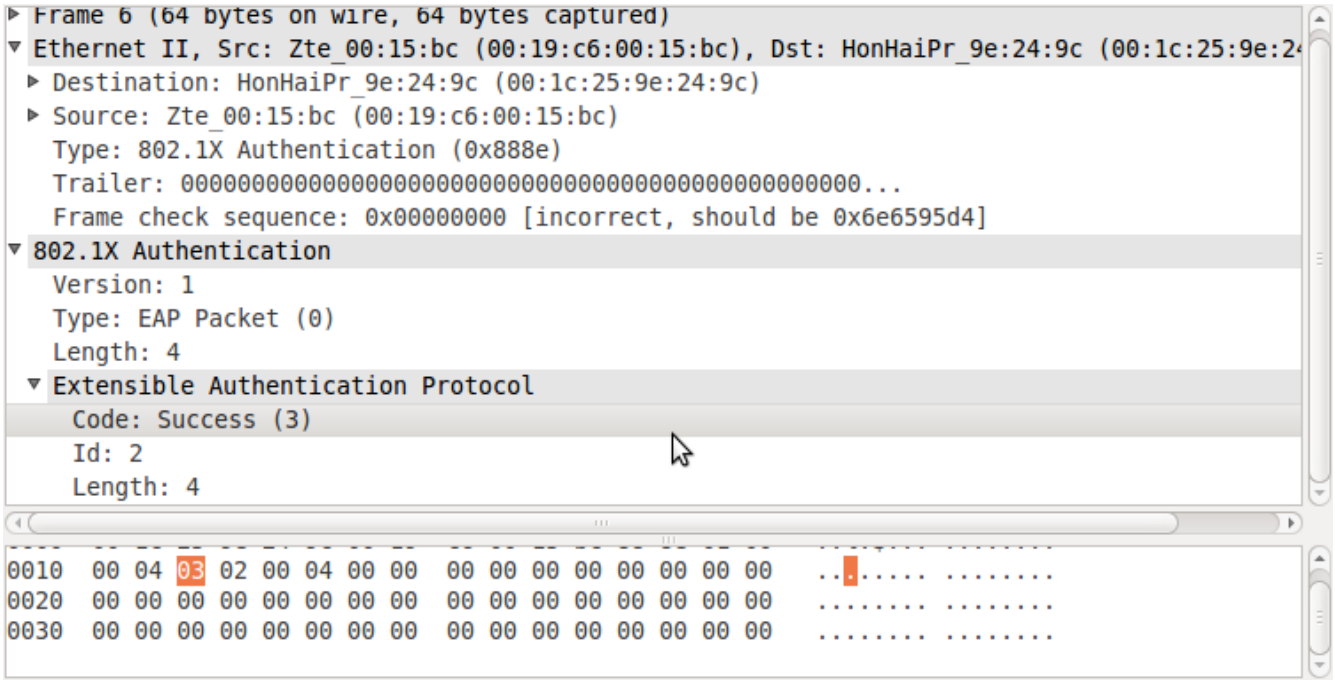


图 2-54 EAP——Success（*Authenticator 发送）

6. Authenticator 將 Supplicant 的 MD5-Challenge Response 发送给 Authentication Server 辨明真伪，Authentication Server 使用 MD5 算法对相同的素材计算，和 Response 值中的 Value 进行对比，如果一致就回应 Success，说明用户可以正常连接入网络了，如图 2-54 所示：

   1) Code 值是 3，表示 EAP 认证结果是 Success，看到这个字样，是一个值得 Supplicant 庆祝的事情；

   2) Identity 依然是 2，看来这个厂家的实现只有 Request 消息会有递增；

   3) Length 值是 4，说明没有 Type 和 Type-Data 字段了。

从这之后，Authenticator 要每隔 10s 检查一次 Supplicant 是否依然活跃（有点类似于 PPP 的 LCP Echo 机制），所以在图 2-48 中发现，0.047s 时收到 Success 消息，到 9.787s 时，Authenticator 又一次发送 Identity Request（不是 MD5-Challenge Request），Supplicant 回以 Identity Response，Authenticator 再此回应 Success，表示本次检查是 OK 的，到了 21.787s 时，这种检查过程又重复了一次。到了 24.783s 时，Supplicant 主动离开网络了，主动离开是通过 802.1X 发送 Logoff 消息：

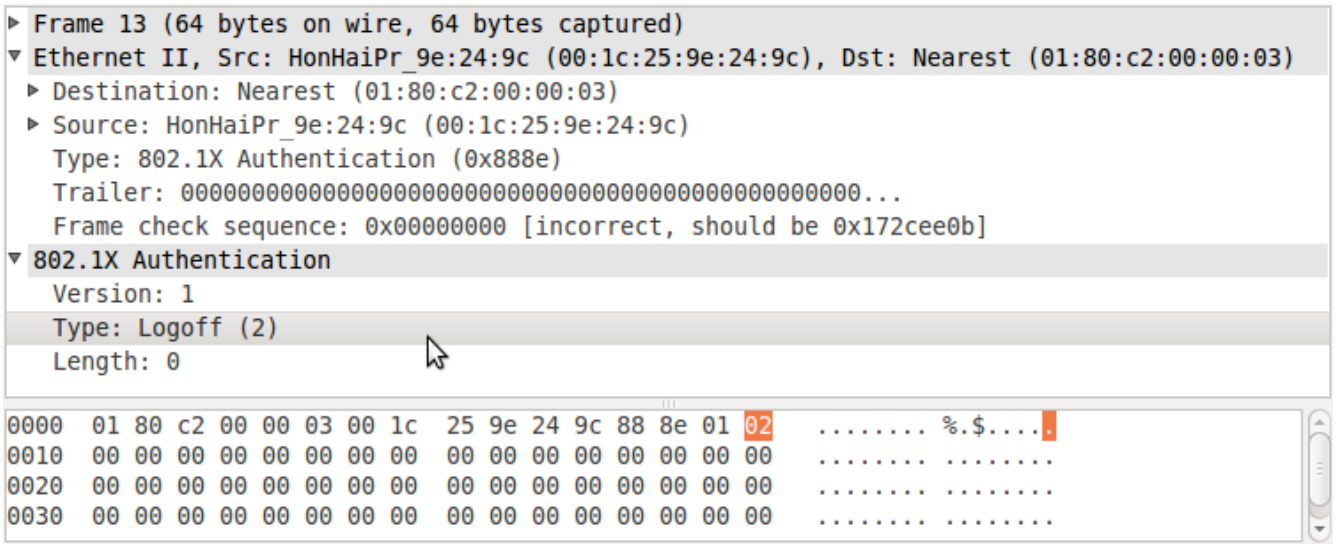


图 2-55 EAPOL——Logoff（*Supplicant 发送）

7. Logoff 是 EAPOL，也就是 802.1X 消息，并不是 EAP 消息，EAP 消息种类只有 Request、Response、Success、Failure 四类，每类消息通过携带不同的 Type-Data：

   1) 802.1X 中 Type 的值是 2，表示 Logoff；

   2) Length 部分是 0，说明后面没有携带任何数据了。

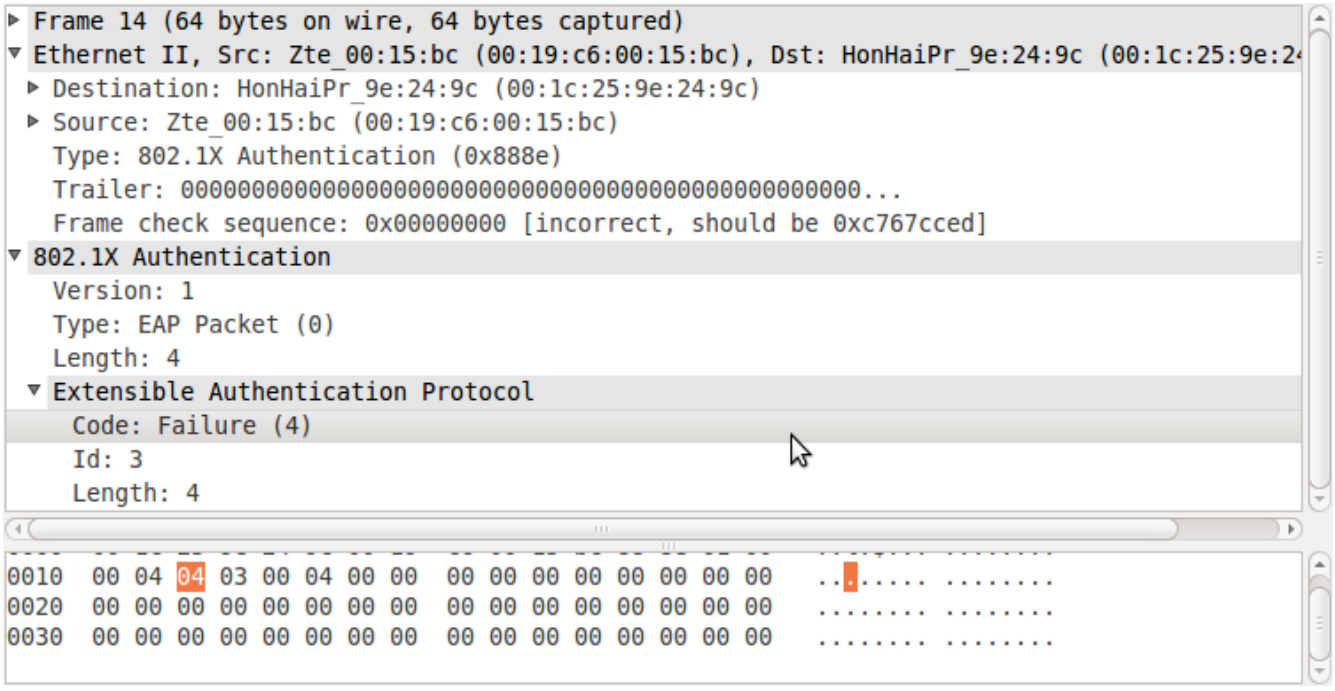


图 2-56 EAP——Failure（*Authenticator 发送）

8. Authenticator 收到 Logoff 后，回应的并不是 EAPOL 消息，而是 EAP 消息，从图 2-56 也可发现 EAP Code 值是 4，表示 Failure，即失败，意思是 Supplicant 不能再连接到网络了，必须重新认证，

Failure 消息的 Identity 是 3，根据观察，9.787s 和 21.787s 的 Request 使用 Identity 都是 3，说明在探测阶段 Identity 就不增长了，只有第一次 Request 会有所增长。

可以发现 EAP Request、Success、Failure 消息都是由 Authenticator 发送给 Supplicant 的，而 Supplicant 发送给 Authenticator 的 EAP 消息只有 Response。实际上，定义 EAP 的 RFC 3748 的确是这么定义的。

上面介绍的是使用 MD5-Challenge 作为 EAP Method 的交互过程，而使用其余 Method 必然有所不同，但基本原理是类似的，Supplicant 需要以某种 Method 向 Authentication Server 提交身份信息以及身份保护信息，Authentication Server 根据这些信息和数据库中保存的资料使用同样的 Method 进行比对，根据比对结果向 Authenticator 下达开放网络连接的许可。再有的网络（主要是共享链路介质型，如 802.11）中，获得许可后还要在 Authenticator 和 Supplicant 之间协商链路层安全密钥，对所有数据帧进行加密，这部分内容会在下一节介绍。

EAP 是 IETF 比较推荐的链路层协议无关认证框架协议，既可以运行在 PPP 链路上，也可以允许在 802 链路上，除了链路层协议无关外，它还可以灵活地支持各种认证方法，但目前 EAP 在 PPP 链路上推广做得并不够，因为 EAP 相比于 PAP、CHAP 优势在于支持新的认证方法，如 EAP-SIM/AKA，但这种认证方法需要得到运营商的支持（支持 EAP 也意味着要大量更换不支持 EAP 的设备，运营商花每一分钱都需要仔细计算是否能加倍地赚回来），目前运营商对于使用 EAP-SIM/AKA 的试验也处于刚刚起步阶段，估计 EAP 会慢慢地部分取代 PAP、CHAP。所以 EAP 运行在 802 链路上的 802.1X 称为 EAP 最主要的实验田，802.1X 的主要市场在于非运营网络（企业 IT 基础设施），通常的方案是將 Supplicant 分为不同的类别，根据用户类别授予不同访问网络的权限，以实现灵活控制接入用户的目的，所以往往 802.1X 设备厂家都会提供一个 Guest VLAN 的特性，指的是如果 802.1X 认证失败，Supplicant 并非访问不了任何网络，而是被动态地划分到 Guest VLAN 中，可以访问 Guest VLAN 内的网络资源（通常是为了寻求认证通过的帮助资源），同时和认证通过 Supplicant 访问的网络资源相隔离，增强了 802.1X 在部署时的灵活度。不过如本节开头所描述，国内大部分企业客户愿意使用接入设备无关的 Portal（通常是不同厂家私有，无法相互兼容）来做网络接入控制解决方案，802.1X 在国外市场会更红火一些，国外数量众多的中小客户的普遍观点是要使用开放标准的解决方案，厂家之间不兼容的解决方案只有在迫不得已的情况下才会采用。国外一个评价开放标准成熟与否的标准就是看看网络上是否有开放源代码（*Open Source*）或者是自由软件（*Free Software*）的实现（摘自 802.11 Wireless Networks: The Definitive Guide, Mathew S. Gast），而中国市场对开放源代码的保护相对来说是不够的。

2.13. 802.11i 与 WLAN 安全

802.1X 提供了一种通过网络接入层设备对 Supplicant 提供网络层访问端口的开关，这种解决方案在以太网

上基本上就很够用了，那么在 WLAN 上够不够呢？答案是不够的，WLAN 的介质是开放在空气中的，任何 WLAN 设备都可以监听介质，也就是说可以监听这个介质上的所有通信内容，这听起来似乎就有点起鸡皮疙瘩，没错，本来是两个人在悄悄话，现在所有的人都能听见悄悄话的内容，这的确令人很不爽。为什么以太网就不会有这个问题呢？以太网在 Switch 面世之前，比如使用 HUB 联网的时候，也是这种状况，只不过当时大家觉得能上网就很是一件很了不起的事情，可以拿出来 Show，所以没什么人 Care 这个隐私问题，而 Switch 的出现，使以太网连接从共享介质变成了点到点介质，通信内容通常不会泄漏到另外一个点到点介质上（通常 Switch 上还可以应用 Mirror 镜像功能，可以将一个介质上的所有内容复制到另外一条介质上，但 Mirror 功能属于管理功能，需要管理员控制，并非人人享有的特权），以太网内容监听问题就告一段落了。本节的主要内容就是介绍一下 WLAN 的安全问题。

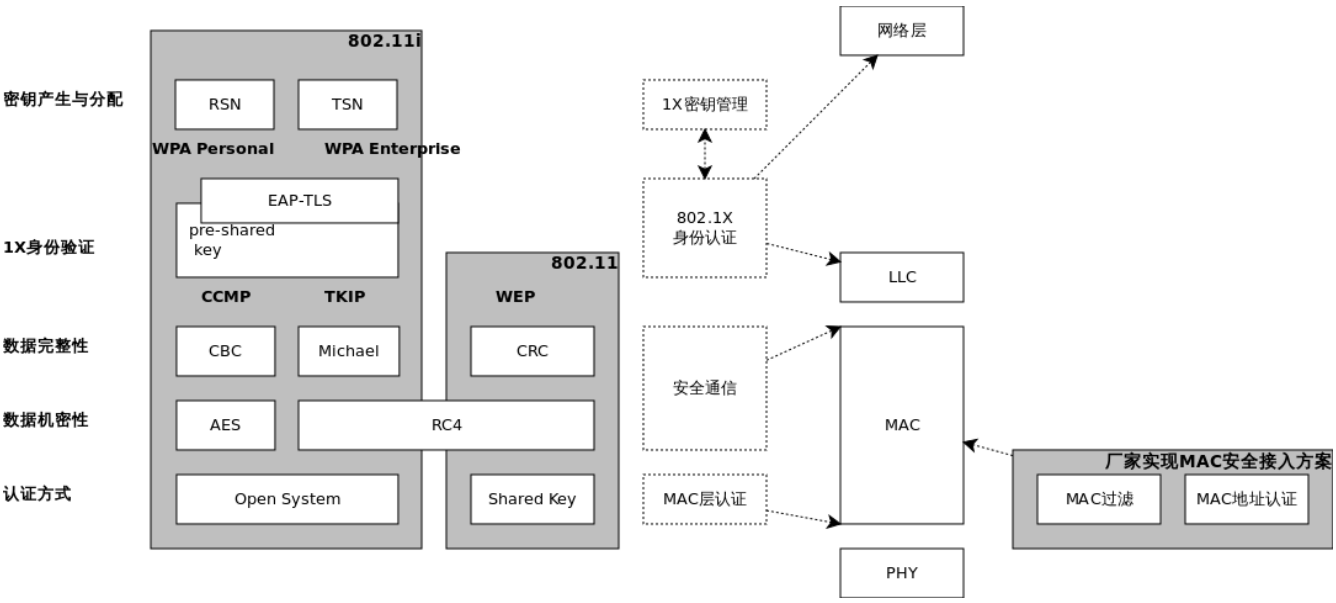


图 2-57 WLAN 安全架构

如图 2-57 所示，其实每次看上面的图，我都感觉到不太满意，因为 WLAN 的安全包含太多的名词，有的名词是大家熟知的，但是另外一个基本技术名词的马甲，要把所有东西介绍清楚不是一件容易的事，还是尝试归纳一下 WLAN 的安全目标：

- MAC 层的“身份认证”，只有“认证”完成才允许 MAC 层互通；
  - Open System，通常被称为开放系统认证，其实就是 STA 之间或者 STA 向 AP 表明自己的 MAC 地址信息，然后就可以 MAC 互通了，这种方式其实不能叫身份认证，应该叫身份展示，因为它并没有任何身份确认信息交互；
  - Shared Key，通常被称为共享密钥，在 Open System 的基础上增加身份保密信息，即所有 STA 之间或 STA 与 AP 共享一个密钥，密钥一直情况下才允许 MAC 互通；目前只有 802.11 早期定义的原生安全体系 WEP 使用 Shared Key，而且其实现方式非常容易被破解，所以 802.11i（Wi-Fi 联盟根据 2003 年 802.11i 中稿推出了 WPA 营销标准、根据 2004 年 802.11i 终稿推出了 WPA2，也就是说 WPA 和 WPA2 都是 Wi-Fi 推出的不同套餐，具体标准都由 802.11

即 802.11i 定义）则使用 Open System 方式，通过 802.1X 来完成身份认证；

- MAC 地址过滤，其实 MAC 地址过滤并不在标准之中，只是厂家在实现时增添的一种安全策略。

- MAC 层的安全通信，凡是安全通信，通常包含：
  - 数据机密性（confidentiality），采用公开的算法（algorithm）、保密的密钥（key）对开放的信息块加密，在密码学 *Cryptographic* 理论中，明文 *clear text* 指的是未加密的信息，密文 *encrypted text* 或者 *cipher text* 指的是加密后的信息）处理：假设加密算法为 Fea，加密密钥为 key1，对应解密算法为 Fda，解密密钥为 key2，ct 表示明文，et 表示密文；加密过程 et = Fea(key1, ct)，解密过程 ct = Fda(key2, et)；从上述公式中，密码学家需要设计出足够安全的 Fea 和 Fda（key1 若和 key2 相同称为对等加密，否则称为非对等加密），使在介质中传递的密文即使被捕获，在没有密钥的情况下难以破译（密码学中破译不止是针对算法的破译，还包括绕开算法直接获取密钥）出明文，所以数据机密性的关键通常是保证密钥的安全和避免使用有规律的明文，WLAN 中使用的加密算法主要有 *RC4*（*Rivest Cipher 4*，RSA 公司 Ron Rivest 设计）和 *AES*（*Advanced Encryption Standard* 高级加密标准），802.11 中的 WEP 使用 RC4 加密，802.11i 中的 *TKIP*（*Temporal Key Integrity Protocol* 临时密钥完整性协议）安全套件也使用 RC4，和 WEP 的区别在于密钥的生成、明文块的构成更加合理，可以说 RC4 是证明足够健壮的，但 WEP 机制比较弱，但许多早期的 WLAN 网卡 MAC 层只支持 RC4 算法，所以 TKIP 是 802.11i 为了兼容老硬件而推出的改良版本，RC4 也受 WEP 的拖累，其安全性受到了质疑，802.11i 最终方案 *CCMP*（*Counter Mode with CBC-MAC Protocol* 计数器模式及密码块链消息认证码协议）安全套件采取了被公认最为硬朗的 AES 算法，WPA 套餐使用 TKIP，WPA2 套餐使用 CCMP；
  - 数据完整性（Integrity）和真实性（Authentication）通常是一起实现，完整性是防止信息被篡改，真实性则防止数据被伪造，WEP 使用 CRC 在密码学上已经被证明是很弱的实现，因此 802.11i 中的 TKIP 套件使用 Michael 方法，而 CCMP 套件则使用更安全的 CBC；
  - 防重放（Anti-Reply），通过监控序列号的方式，防止收到过期的数据帧，过期的数据往往被认为是一种攻击，TKIP 和 CCMP 都实现了，但防重放攻击检测和 802.11e 中的 QoS 有一部分的冲突，需要特殊处理。

- 802.1X 可以使用多种 EAP 身份认证方式：
  - pre-shared key 方式被称为“预共享”密钥，它并非标准的 802.1X 身份认证，而是用于 802.1X 协商密钥，协商密钥时使用 pre-shared key 进行身份确认（只有一致才能协商出最后的密钥），即使所有 STA、AP 使用相同的 pre-shared key，但密钥协商是独立的，即每个 STA 与 AP 协商出来的密钥都是不一致的，可以确保通信安全，WPA Personal（个人级）套餐使用 pre-shared key 认证方式，有时它也会被简称为 WPA-PSK；

    - EAP-TLS 方式，即使用数字证书做认证，WPA Enterprise（企业级）套餐使用该方式，因为 TLS 方式部署比较困难，所以有的 WPA Enterprise 会使用 PEAP（Cisco 主推）或者 TTLS（Microsoft 主推）方式替代；
    - 还有的厂家会使用 MAC 地址认证来取代 802.1X 认证，MAC 地址认证类似于 MAC 地址过滤，方式，是一种策略，并非协议标准；
- 在介绍 802.1X 时还提到 802.1X 在身份确认后还可以提供密钥交互过程，802.11i 中的密钥在使用 802.1X 交互之后有两种密钥管理和分配机制：
    - *RSN*（*Robust Security Network* 强壮安全网络）定义了一种层次化的密钥产生与分配机制，已经是目前时尚的方案；
    - *TSN*（*Transition Security Network* 过渡安全网络）和 TKIP、CCMP 的关系一样，在 802.11i 尚未终稿时的密钥产生、分配机制被称为 TSN，802.11i 终稿后的标准就是 RSN。

WLAN 的安全框架是依托于 802.11（制定了早期的 WEP 安全套件）和 802.11i（制定 TKIP、CCMP 安全套件和 RSN 密钥管理体系）标准建立的，由于时间跨度过长、WLAN 推广较快的原因，使 Wi-Fi 产业联盟在制定营销策略时不得不考虑兼容已有老的硬件，最终的结果就是推出了许多名词如、许多标准（如 WPA、WPA2、WPA Personal、WPA Enterprise）。WLAN 安全框架中的那么多名词在实际网络部署中都很容易遇到，而且在部署中是很灵活的组合方式，比如 WEP 方式下也可以使用 Open System 认证，但如果密码不对，依然无法连接到无线网络，希望通过这幅图能够让大家在网络部署时可以更清晰地选择最合适的组合，估计再过若干年，网络就慢慢地都切换到 802.11i 终稿所定义的 WPA2 了。在这里，我们还是展示一下被加密的 802.11 数据帧：

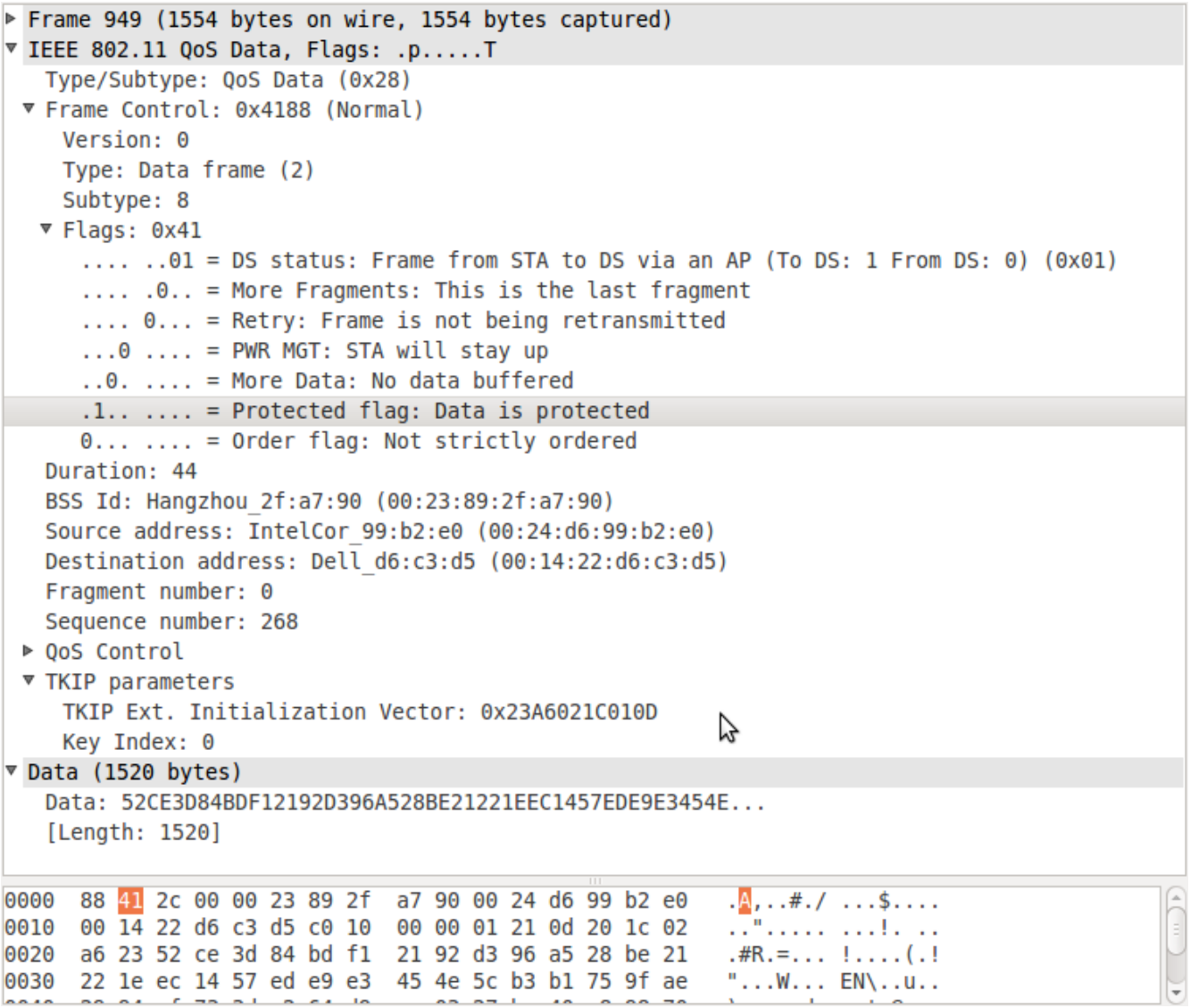


图 2-58 802.11 加密数据帧

可以从帧的结果分析数据帧的 Protected 位被置 1，表示数据帧已经被加密了，加密套件使用 TKIP，Data 部分长度为 1520 字节。那么怎么判断使用 TKIP 套件呢，这是在 MAC 层协商的结果，请看 802.11 MAC 层的协商过程：

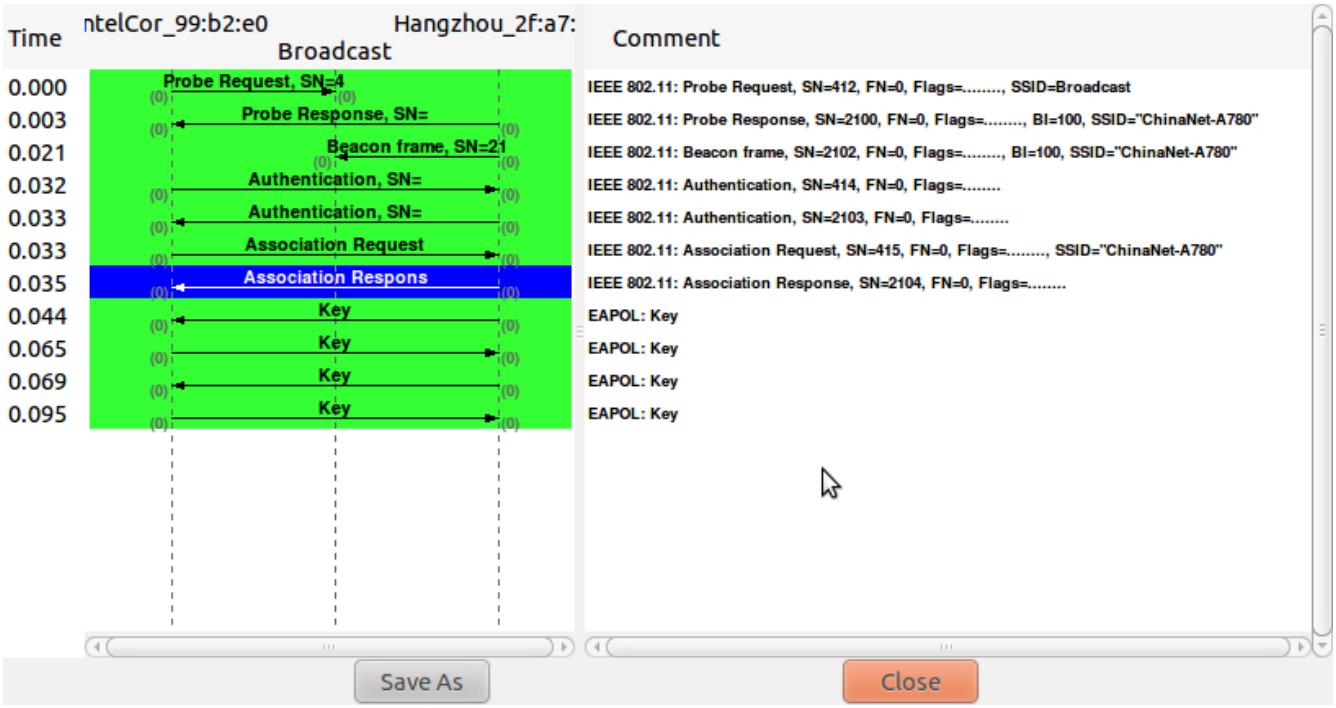


图 2-60 802.11 MAC 层协商与 802.1X 协商密钥

如图 2-60 所示，可以把整个过程分为 4 部分：

1. 扫描 scanning 阶段，即 STA 主动扫描所有的 AP：
   1) 如第 1 个帧所示，STA 发送一个 probe request（SN 号为 402）给 Broadcast；
   2) 随即 AP 回应第 2 个帧 probe response（SN 号为 2100），该帧除了携带表明该 AP 所提供服务名称（术语为 Service Set ID，简称 SSID）为 ChinaNet-A780 外，还携带 AP 所支持的安全套件，该帧需要肯定确认，即等待 Ack 回应；
   3) STA 发送 probe request 查询 AP 方式称为主动扫描，如果 STA 监听 AP 周期发送的 Beacon 发现 AP 则称为被动扫描，第 3 帧就是 AP 周期性发送的 Beacon（SN 号为 2102），STA 可能会在一段时间内发送若干个 probe request，AP 会逐个响应，SN 为 2101 帧就是如此，故在此删除；
   4) Ack 帧并未在 Wireshark 流程图中显示。
2. 身份“认证”authentication 阶段，其实叫身份展示更合适，因为 WPA 使用 Open System 方式：
   1) 第 4 帧为 STA 发送的 authentication，表明 STA 使用 Open System 方式，没有任何身份信息；
   2) 第 5 帧为 AP 向 STA 发送的 authentication，内容和第 4 帧一致。
3. 关联 association 阶段，association 阶段是 MAC 层协商的最后一步：
   1) 第 6 帧为 STA 向 AP 发送的 association request，里面携带着 WLAN 的一些物理层参数和所支持的加密套件如 TKIP、pre-shared key、RSNv1 等等；
   2) 第 7 帧为 AP 对 STA 的确认 association response，是对 STA 请求参数的确认，当 STA 收到此帧后要给予 Ack 应答，也就表明 STA 和 AP 之间的 MAC 层通信被打开，可以进行 LLC 以上层

的通信了。

4. 802.1X 是基于 LLC 层的协议，所以只有 MAC 层通信打开后才能进行：
    1) 第 8 帧到第 11 帧是 802.1X 使用 pre-shared key 协商密钥的过程，密钥协商完毕后就会被用于 TKIP 进行数据机密性、完整性、真实性的实现，如果 pre-shared key 不一致则协商失败，重新协商，多次失败后 STA 的实现可能会从 MAC 层重新开始。

在了解到 WLAN 的安全框架后，我们很容易可以将 WLAN 的安全问题进行归类：

1. 不使用任何安全措施的 WLAN 也是允许的，但往往是为了实验；
2. WEP 类型的 WLAN 要保护 shared key 被破解，否则任何未经授权的 STA 可以接入网络，但 WEP 这方面非常弱，很容易被破解，市面上存在许多许多蹭网卡；
3. WPA-PSK 类型 WLAN 要保护 pre-shared key 被破解，否则后果与 WEP 一样，但目前破解 WPA-PSK 还是比较困难的；
4. WPA Enterprise 是非常安全的，证书体系是目前为数不多值得信赖的方案；
5. 防止加密过的数据帧（WEP、WPA 均是加密传送）被破译；
6. 防止重放攻击和拒绝服务攻击。

由于涉及密码学的过多知识，本人精力和能力都为有限，在这里并不详细介绍每种算法如何实现，着重于介绍流程和框架，如果读者对此非常感兴趣，可以参考 Matthew S. Gast 的《802.11 Wireless Networks: The Definitive Guide》（中文译名为 802.11 无线网络权威指南）及密码学、IEEE 相关文档。

2.14. 802.11e 与 WLAN QoS

我经常在想，网络中的 QoS 和现实生活有没有什么联系，有一些东西先和大家分享：

- 食堂排队或者银行什么的排队大家都遇到过，想想哪个方式最高效又兼顾公平：
    - 单窗口不排队竞争制；
    - 单窗口单队列，分别考虑有人插队和没人插队的情况；
    - 多窗口多队列，考虑如何选择队列，以确保自己最快被服务；
    - 多窗口多对列，考虑如果某个队列是 VIP 队列的情况；
    - 多窗口单队列，但每个人需要领号，等待被叫号，考虑有些号是 VIP 号，考虑有些号人已经走了。
- 某天某地方某大街限行，是为了让官员的专车畅通，我们可以考虑一下怎么辨别这辆车是民车还是官车?
- 一个十字路口，其中一条路是小路，而其它 3 条大路上的车都想往小路上开，是不是必然会发生拥堵?
- 如果窗口比排队的人还多，那还需不需要排队呢？如果道路一点都不堵车，是不是还需要交通管制?
- 在一个 50 人教室自由早读课里要相互交流即使声音很大也非常吃力，但如果是老师点读或领读，是不是很多人会想着怎么样说说悄悄话?

- 我突然又想到一个现象，2 年前逛街，手里拿一 iPhone 是不是感觉特骄傲，即使移动、联通服务很差，再对比一下现在，大街上人手一个 iPhone，那种 show 的想法荡然无存，开始在意移动、联通的服务了?

是不是有点意思，服务质量（Quality of Service）是网络界过目率比较高的词汇，但和艺术来源于生活类似，网络 QoS 经常在人类社会中找得到原型：

- 什么时候需要 QoS，当网络资源不够时我们会在意服务质量，它的起源往往是人们的心理诉求，就比如上面路面不堵车和 iPhone 变成街机的例子都是如此；
- QoS 要解决什么样的问题，在当前资源条件下，提高资源利用率又兼顾相对公平，和交通管理是一个道理，在变得拥堵的路面上，如何通过更多的车辆，又要照顾不同方向的司机，甚至要为一些特殊车辆预留一些道路，又比如教室自由早读问题，可以理解成如何能让更多的人清晰交流，而不是相互叫嚷；
- QoS 的基础是什么，分类、排队、调度资源，这和我们说的食堂排队是一样的。

我们常说的 QoS 是网络技术的统称，具体来说又可以分为下图所描述的一样：

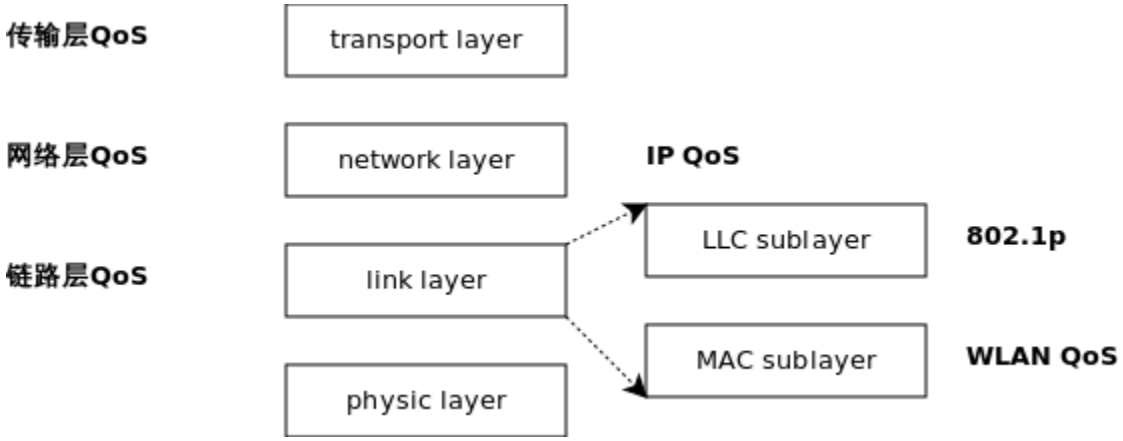


图 2-61 TCP/IP QoS 体系架构

可以看到 TCP/IP 中能够部署 QoS 的地方还真不少，通常部署在链路层和网络层的比较多，它们的意图如下：

- 网络层 QoS 技术面向 end to end 的数据传送，通常会將网络层 QoS 按照传输路径映射到不同的链路层 QoS 技术实现，比如官员要求到不同地方乘坐当地的专车都要享受优待，官员的要求就相当于网络层 QoS，他乘坐的地方专车就是链路层 QoS，因为地方交管只认专车，而不认专车里坐的是哪位高官，但是不是每个地方享受的优待都是同样标准都一样就不好说了，说不定换个地方有更高的官，所以网络层 QoS 只是一种对 end to end 服务的诉求，最终的落实要靠链路层 QoS，现实的情况是，硬件已经將网络层 QoS 和链路层 QoS（具体说是 LLC 子层）一视同仁，区别并不大，真正的差异在于 MAC 层 QoS；
- 链路层 QoS 解决链路上传输数据问题，也就是实质上的技术，802 网络又把它细分为 2 部分：
    - LLC 子层技术如 802.1p，用于解决逻辑链路上 transmitter 到 receiver 的问题，特别是网桥上，因为网桥就相当于链路层的十字路口，是容易发生拥堵的地方，以太网交换机是目前最流行的网

桥，所以 802.1p 在以太网交换机上应用极广，这也是为何网络层 QoS 可以和 LLC 子层 QoS 联合的原因，因为网络层也是链路的边界，也是一种链路的十字路口，既然都是十字路口技术，虽然层次不一样，但可以一并给实现了；

- MAC 子层技术如 802.11e 专门用在 WLAN 共享介质，它解决的就不是十字路口拥堵了，而是解决教室自由早读问题，WLAN 的介质是天生开放的，大家自由地朗读，导致谁也听不太清楚谁，，WLAN 中专业的说法是 *contention* 竞争，所以这里存在冲突几率和介质利用效率的问题，WLAN 领域中有一个重要的挑战就是如何解决密集环境中的网络可用性问题，比如在拥有 2 000 个 STA 的苹果开发者大会，乔老爷的演示经常因为 WLAN 网络过于繁忙变得极不流畅，会前往往会呼吁与会者关掉 STA，另外一方面每年的麦加朝圣，更加密集的信徒们依然可以使用手机发送短信，WLAN 若不能解决密集使用网络问题，其吸引力会打折，解决这个问题主要靠开源和节流，所谓开源就是提高介质速率，如换成更高速的 802.11n，但 STA 一多依然会遇到瓶颈，所以还需要节流，使介质使用更加高效，802.11e 就是一种节流技术；以太网自从交换机问世后，使用的是点到点介质（如图 2-3 所示），也就是专享介质，并不存在多点冲突，以太网的 MAC 层 QoS 技术因此而很少见，同样 PPP 链路是点到点专享链路所以连 PPP 链路层 QoS 也很少耳闻（PPP 链路 QoS 往往借助于链路边界 router 网络层 QoS 实现），假设以太网目前还不幸地停留在 HUB 时代（如图 2-2 所示），我认为以太网的 MAC 层技术也会很红，但这只是假设，宁愿它不曾发生；MAC 层 QoS 技术到了介质边界如网桥基本上就不会再到下一个介质生效了，在众多 QoS 技术中，解决冲突提高效率的 MAC 层 QoS 最为复杂，但也最少接触，因为都是通过硬件芯片实现，使用者甚至不需要知道，这次介绍的 802.11e 就是如此。

那么 802.11e 是如何做到这些的呢？WLAN 使用 CSMA/CA，但一般来说 CSMA/CA 工作在物理层，802.11 在 MAC 层 *Virtual Carrier Sense* 虚拟载波监听使 CSMA/CA 机制更加灵活和有效，虚拟载波监听使用 *NAV*（*Network Allocation Vector* 网络分配向量）分配不同 STA 可以占用介质的时间，NAV 的具体表现就是图 2-7 中的 Duration 字段。MAC 层发送数据帧的判断因素称为访问方式，WLAN 的访问方式如下所示：

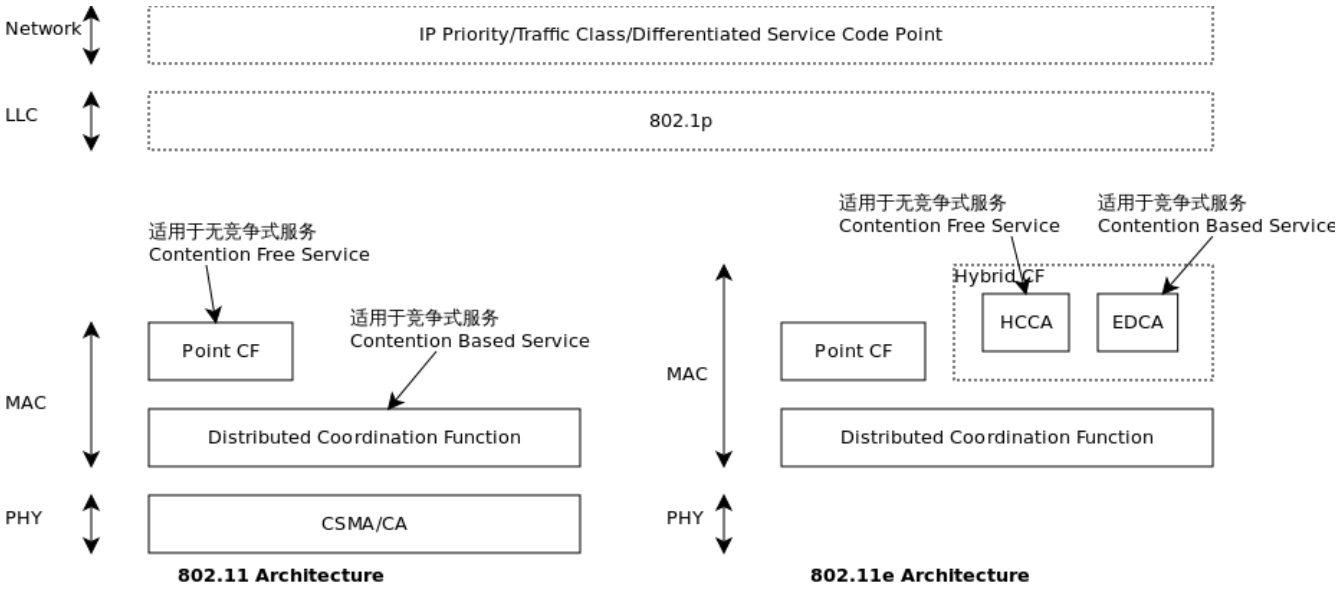


图 2-62 802.11 MAC 访问体系结构

在原生的 802.11 中 MAC 访问方式主要有 2 种方式：

1. 基于竞争的 *DCF*（*Distribute Coordination Function* 分布式协调功能），所有的 STA 及 AP 监听相同介质或信道上任何数据帧的 Duration，根据 Duration 判断该介质何时空闲，在介质空闲段内等待随机的 *back-off* 退避时间后将数据帧交给 PHY 层，容易想象如果 2 个 STA 随机的 back-off 时间恰巧相同，那么就很容易发生介质占用冲突，双方只能重传，因此这种使用方式是有竞争的；
2. 无竞争的 *PCF*（点协调功能），DCF 中是所有 STA 和 AP 各自维护介质使用权，因此是有竞争的，PCF 将介质使用权完全交给 *PC*(*Point Coordinator* 点协调单元)，通常是 AP，由 AP 来决定 STA 的介质使用权，在 PCF 中，AP 会定期发送 *CF-Poll* 轮询各个 STA 是否有数据帧发送，CF-Poll 中的 Duration 通常涵盖了 STA 发送数据帧需要占用介质的时间，AP 牢牢地把握着 NAV 的使用权，802.11 默认使用 DCF，Point 可以发送特殊的 Beacon 帧向所有 STA 及 AP 宣告该介质进入到 *CFP*（*Contention Free Period* 无竞争周期），Point 通过发送 CF-End 恢复到 *CP*（*Contention Period* 竞争周期）。

802.11 中定义的 DCF 和 PCF 都不能完全满足大家对无线网络的期盼，这是因为 DCF 和 PCF 都是基于 STA 进行调度的，而每个 STA 使用网络的应用可能有很多，比如视频、语音、下载，STA 的使用者通常是希望语音和视频得到比较好的服务。这下我们都了解了，WLAN 的用户希望网络资源不仅是基于 STA，能够基于应用调度就更好了，这就是 802.11e 提出的 *HCF*（*Hybrid Coordinated Function* 混合协调功能）：

1. 802.11e 提出了 *HCCA*（*HCF Controlled Access HCF* 控制接入），以改良无竞争模式下的 PCF 表现；
2. 提出了 *EDCA*（*Enhanced Distributed Channel Access* 增强分布式信道访问）改良了竞争模式下 DCF 的表现（DCF 不能区分不同应用）；
3. 从图中也可以看出，802.11e 保留了 PCF 和 DCF，因而可以兼容 802.11，支持 802.11e 的 STA 被

称为 QSTA，同样有 QAP、QBSS 的概念，HCF 中的协调单元通常就是 QAP 也被称为 *HC*（*Hybrid Coordinator* 混合协调单元）。

要看这个带 e 的玩意怎么比不带 e 有所增强，还是需要对比一下具体的变化：

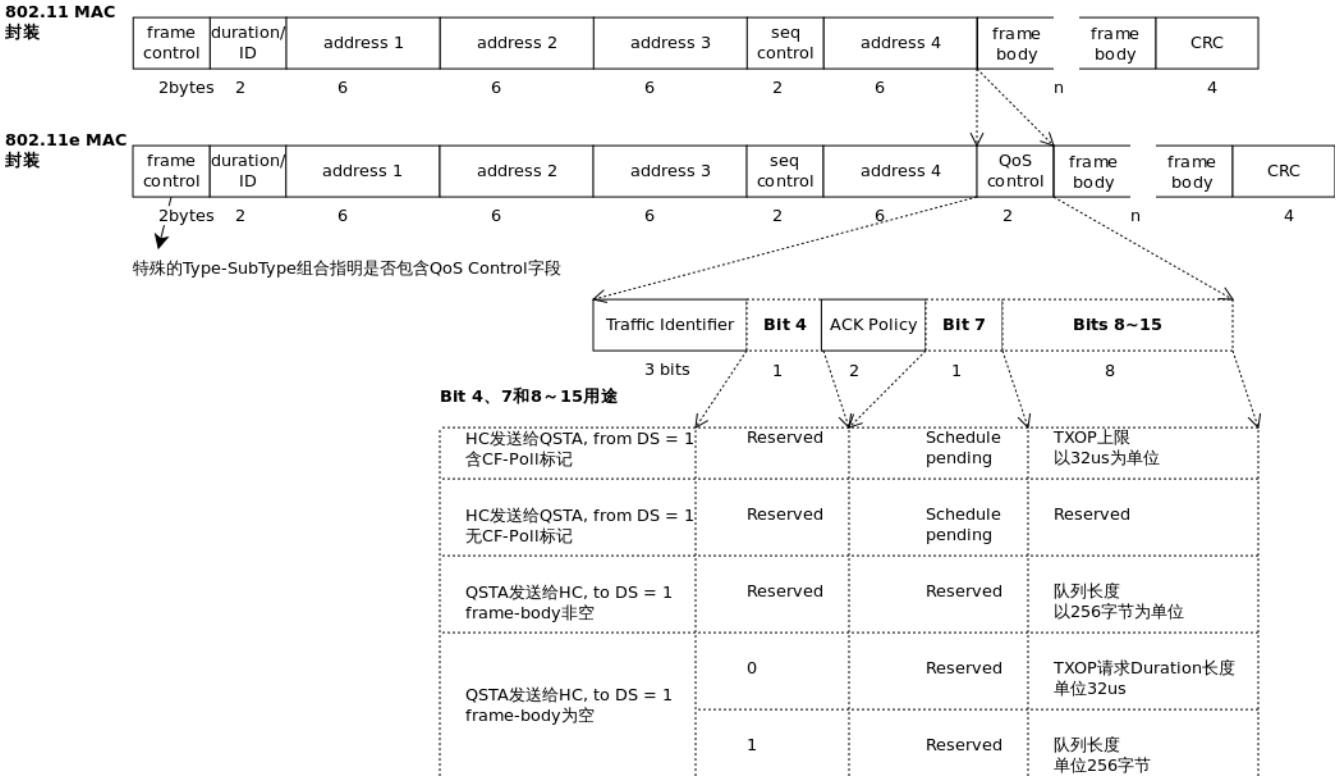


图 2-63 802.11e 帧格式变化及流分类

从图 2-63 看出，在图 2-7 中的 802.11 封装其实是 802.11e 封装，目前正常运行的 WLAN 基本上是 802.11e 封装，但未必就真正实现了 802.11e，图 2-63 也标出了原生的 802.11 封装的模样，那么怎么区分 802.11e 封装呢，就看 Type-SubType 的组合了，目前 SubType（长度 4 位）在 1000 以上的就是，802.11e 封装也被称为 QoS Frame，802.11e 封装多出部分就是 2 字节的 QoS Control：

1. 3bit 长的 *Traffic Identifier* 流标识 *TID*，它在数值上和 802.1p 保持一致（数值越大优越感越强），但使用方式有所不同（802.11e 根据 AC 制定策略，而非 TID），如下表所示：

TID 或 **802.1p**	**802.1p** 描述	**AC**（***Access Category*** 访问分类）	**802.11e** 描述
0	Background	0	Best Effort
1	Best Effort	0	Best Effort
2	Excellent Effort	0	Best Effort
3	Critical Apps	1	Video Probe
4	Video（延时小于 100ms）	2	Video
5	Voice（延时小于 10ms）	2	Video
6	Internetwork Control	3	Voice
7	Network Control	3	Voice

2. 1bit 长的第 4 位，该位为 0 时（reserved 也是 0）表示使用 *Prioritized* 优先级方式 QoS，该位为 1 时表示使用 *Parameterized* 参数化 QoS，二者的区别如下：
    1) 优先级方式 QoS 根据优先级制定一种相对的优先策略，比较松散，也被称为 DiffServ 差分服务 QoS；
    2) 参数化 QoS 显得更加精细，因为它需要一些服务参数的约定，比如带宽，比如延时，也被称为 IntServ 集成服务 QoS；
3. 2bit 长的 ACK Policy，802.11 是逐帧确认的，只有前一帧得到确认，才会发下一帧，而 802.11e 则可以组确认（Group ACK），ACK Policy 指的是采取何种确认策略，目前有 3 个值：
    1) 0 表示不需要确认，已提高效率，但牺牲了可靠性传输，可靠性传输由高层如传输层实现；
    2) 1 表示正常确认，即逐帧确认方式，确认帧不包含任何被确认帧信息，所以这种确认被认为是半可靠的；
    3) 2 表示块确认，即确认一连串的数据帧，类似于传输层 TCP 的确认技术，即提高了效率，也没有过多地损失可靠性。
4. 1bit 长的第 7 位，大部分时候被保留，在 HC 发送给 QSTA 方向时（to DS = 0；from DS = 1）用于表明当前队列是否被挂起；
5. 8bit 长的第 8 位～第 15 位，主要有 2 种功能，用于表示 *TXOP*（*Transmission Opportunity* 发送机会），单位 32us；另外就是队列长度，单位 256 字节；当 HC 发送给 QSTA 时表示允许值，QSTA 发送给 HC 时表示请求值。

文字性地掌握了许多名词字段后，我们来通过这幅图了解具体的 MAC 访问：

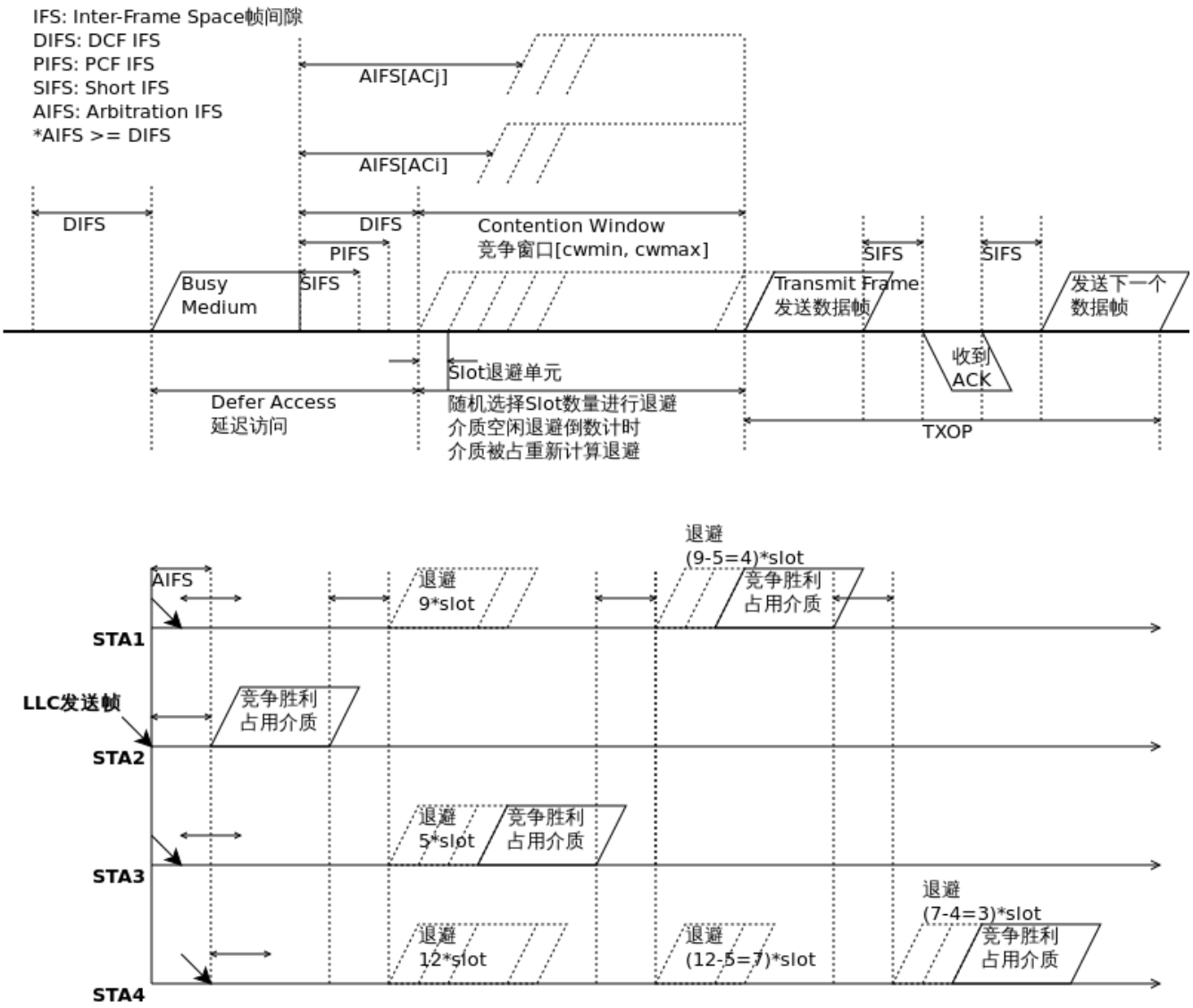


图 2-64 MAC 访问

图 2-64 上半部分，描述了 EDCF 及 DCF 的访问过程：

1. LLC 层將帧交给 MAC 后，MAC 层要等待 NAV 所指使的介质空闲；
2. MAC 层检测到介质空闲后要等待不同的帧间隔，这被称为延迟访问，SIFS<PIFS<DIFS<=AIFS：
   1) SIFS 通常给原子操作使用，如 RTS/CTS，数据帧的 ACK 等；
   2) PIFS 用在无竞争 PCF 中；
   3) DIFS 用在 DCF 中；
   4) AIFS 用在 EDCF 中，其余 3 种 IFS 都是固定值，AIFS 则是动态的，不同的 AC，其 AIFS 是不一样的，所以不同 AC 对应的 AIFS 被用函数表示为 AIFS[ACi]，每个 AC 对应的 AIFS 可以由 AP 来指定，从这里可以看出 HC 有办法让低级别的 AC 等待的 IFS 比其它 AC 长。
3. 当 IFS 等待结束后，如果介质依然空闲，还需要等待竞争窗口 CW（也被称为退避窗口），CW 的单位是 slot，每个 slot 的时间是固定的，根据物理层类型而定，CW 通常用范围[cwmin, cwmax]表

示，不同 AC 对应的 cwmin 和 cwmax 也是由 HC 来指定，在发送数据帧时在 CW 窗口中挑选一个随机值 n，然后开始倒数（图 2-64 下半部分展示了 4 个 STA 竞争过程，假设刚开始 STA2 取得了介质使用权先发送数据帧，分析 STA1、STA3 和 STA4）；

4. 如果倒数结束，介质依然空闲，那么就 MAC 层就算竞争胜利，可以发送数据帧如图 2-64 下半部分的 STA3 就是这个幸运儿；
5. 如果倒数没有结束，比如倒数了 m 介质被其余 STA 占用了，那么停止倒数，继续等待介质被释放：
   1) 如果介质空闲，则等待 AIFS；
   2) AIFS 结束继续倒数 CW，但此时 CW 的 slot 数量是 n-m，也就是说比第一次倒数短了不少时间；
   3) 倒数结束介质空闲，则发送数据帧，如 STA1；
   4) 如果还是未能发送则继续重复以上步骤，如 STA4；
   5) 这里没有展示冲突的情况，如果发生冲突则以上步骤需要重新走一遍，所以很多人想把 CW 调成 0 以增强竞争力的想法是不现实的，那只会导致无休止的冲突。
6. 在 EDCA 中还有 TXOP 的概念，TXOP 是由 HC 通过 Beacon 通过给所有 STA 的，所有相同的 AC 享有相同的 TXOP，当 STA 取得介质占用权后就进入到 TXOP 阶段，此时 STA 发送数据帧是突发模式，即帧一旦被确认，只需等待 SIFS 就可以发送下一个数据帧（TXOP 中还可以使用 GACK 进一步提高效率）；在 DCF 中没有 TXOP，意味着每个数据帧发送都需要重复以上的竞争过程。

从以上部分可以看出 EDCF 实现了不同 STA 内部不同 AC 的 MAC 访问时机和介质占用机会是可以通过 HC 来控制的，而不像 DCF 那样，所有人机会均等，关键业务并得不到保证，最坏的情况是所有业务都无法使用：

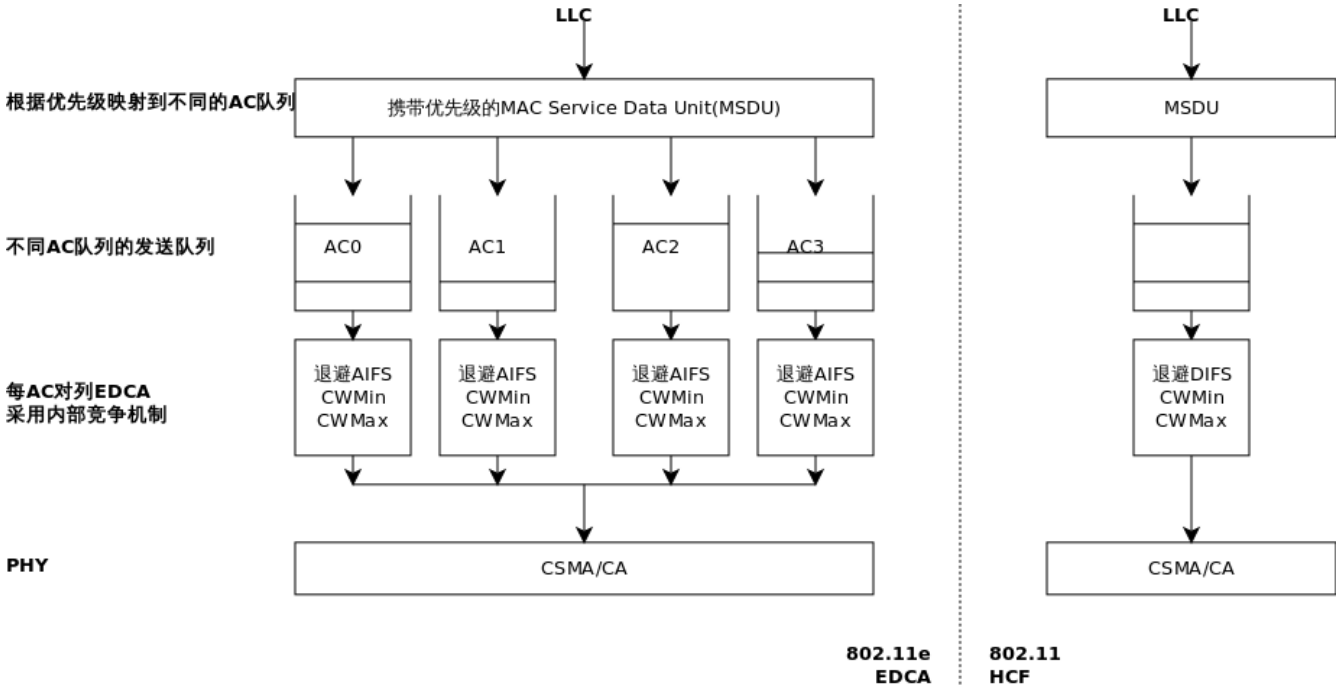


图 2-65 EDCF 和 DCF 队列机制对比

虽然 EDCF 相比于 DCF 有了长足的改进，能够让一些关键的应用在竞争中占优，但依然存在竞争，冲突不可避免，当 STA 数量越多冲突將会越明显。正如 PCF 是 DCF 的补充，HCCA 是 EDCF 的补充，下面就对比一下 HCCA 和 PCF 的差别：

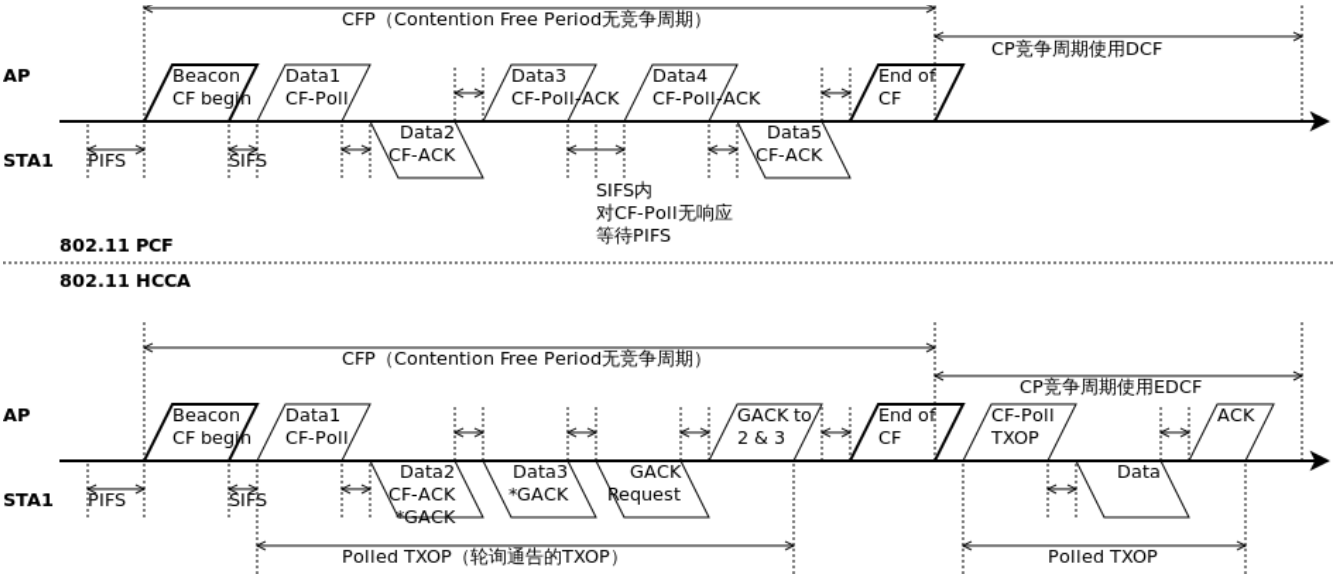


图 2-66 PCF 与 HCCA

在 PCF 中，当 PC 在介质空闲 PIFS 时间后，通过一个特殊的 Beacon 通告介质上所有 STA 进入到 CF 周期：

1. CFP 内大部分数据帧的间隔是 SIFS；
2. PC 每次轮询 1 个 STA，询问其是否有数据发送，其余 STA 不许在这段时间内捣乱，被轮询 STA 只需等待 SIFS 就可以发送数据，的确要比 DCF 的 DIFS 强不少；
3. 如果 PC 在 SIFS 没有收到被轮询 STA 发送的数据，它还需要等待 PIFS，再发起下一个轮询；
4. PC 觉得没有必要再通过轮询来维护介质时，就会发送一个 End of CF 宣告回到竞争周期。

PCF 看起来不错，但依然拥有改进的空间，如没有什么轮询策略（策略粒度仅限于 STA），每次轮询只能发送 1 个数据，那么我们看看 HCCA 做了哪些加强：

1. HCCA 使用 802.11e 帧格式，每个数据都属于特定的 AC，STA 可以和 HC 协商每个 AC 要享有的策略，在策略的制定上拥有了依据，HC 可以通过 CF-Poll 单独向某个 STA 的某个 AC 颁发 TXOP，使策略的粒度细化到每个 STA 的 AC；
2. 可以通过 TXOP 和 Group ACK 的结合，使被轮询的 STA 可以不用每发一个数据帧等 ACK，而是可以在宣告 TXOP 内以 SIFS 的间隔连续发送数据帧，然后请求组确认，以提高效率；
3. 在 CP 阶段，HC 依然可以通过 CF-Poll 向某个 STA 的某个 AC 宣告 TXOP，未被宣告的 STA 仍然使用 EDCF 的 TXOP，这使 QoS 策略变得更加灵活，个人认为 HCCA 用在乔布斯的苹果全球采摘节上可能更合适一些，但总体而言 WLAN 相比于 3G 等运营无线网络的 QoS 还是有一定差距，但 WLAN 的优势是成本、高速、自由，这是我们用它的原因。

从以上描述可以发现 802.11e 定义了一些比较有效的手段，802.11e 还有其余手段如 *DLP*（*Direct Link Protocol* 直接链路协议）、*APSD*（*Automatic Power-Saved Delivery* 自动节能传送）以及一些 TXOP、GACK 的深度分析因为能力和精力的原因不做过多介绍。

802.11 MAC 层技术通常是硬件实现，经常遇到的配置其实并不多，WLAN 相关热点技术还包括节能等话题也未能介绍，WLAN 的知识浩如烟海，《802.11 无线权威指南》一书差不多就和《TCP/IP 详解》卷一差不多厚，可见网络到了底层其实也相当相当复杂，尤其是和无线射频沾边，叙述不周之处还望海涵，描述不对、文法不当之处还望指出。

2.15. Loopback Interface 环回接口

大多数 TCP/IP 协议栈的实现都会提供环回接口，学习知识经常遇到一个学习深度问题“知其然，知其所以然”，环回接口出现的原因我是这么想的，如果 TCP/IP 协议栈一定要通过一个物理上的接口才能工作的话，成本比较高，意味着必须通过一些物理接口连接另外一套协议栈才能让 2 套协议栈工作起来，就像图 1-2 所展示的那样。如果设计一个逻辑上的虚拟物理接口，让它永远 UP，虚拟接口上运行 TCP/IP 协议栈，那么在一台机器内就能进行许多 TCP/IP 应用的测试，这种虚拟的接口就是环回接口：

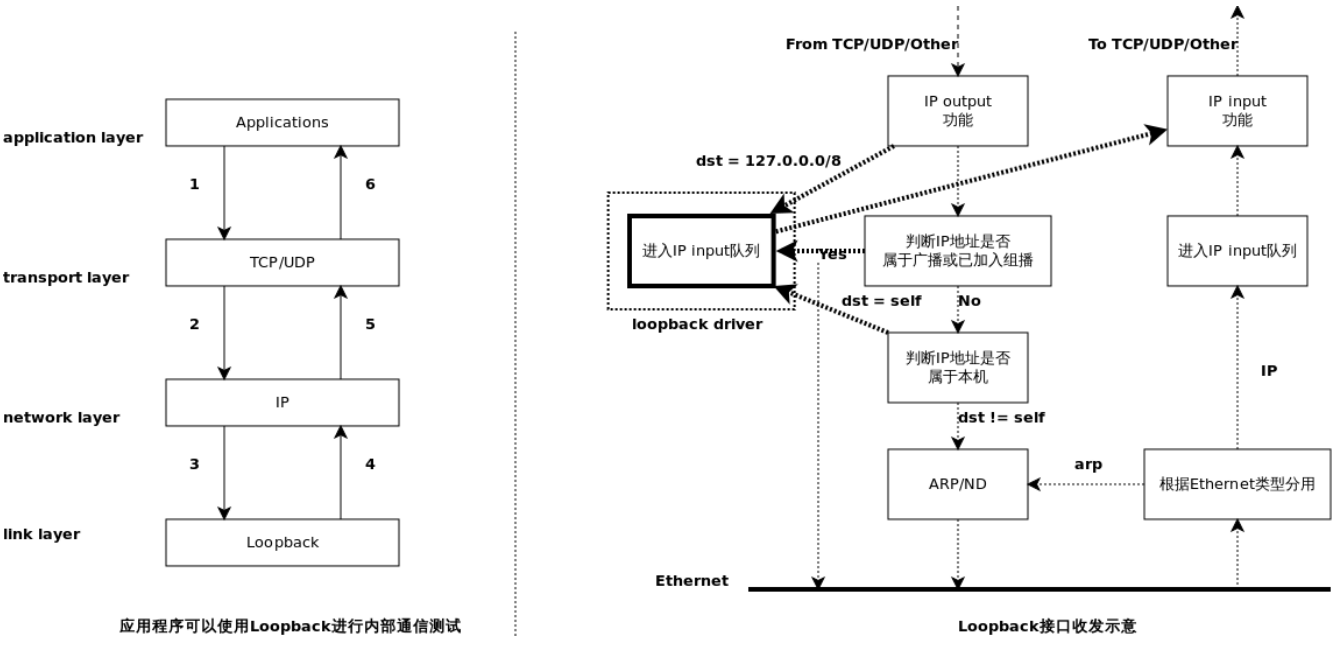


图 2-67 TCP/IP 使用环回接口进行通信

如图 2-67 左侧所示 1～6 步，应用程序可以通过 Loopback 接口完成 TCP/IP 协议栈的完整旅行，那么什么样的流量需要经过 Loopback 呢，可以观察图 2-67 右侧：

1. 环回接口是一个链路层的接口，从图 2-67 来看，它的输入只能是来自本地的 output 功能（并不可能来自物理接口，当物理接口状态 Up 时，该物理接口配置的 IP 地址就存在，否则就不存在），可以分为 3 种：

1) 目的 IP 地址是 IPv4 的 127.0.0.0/8 或者 IPv6 的::1/128，这两个地址分别是 IPv4 和 IPv6 分配给环回接口使用的环回接口地址；

2) 目的 IP 地址是广播地址或者是本地已经加入的组播地址；

3) 目的 IP 地址是所有本地各类接口的地址。

2. 它的输出只能是本地的 IP input 功能，这就很形象的解释了为什么叫“环回”——流量从本地的 output 都回到本地的 input。

环回接口因为始终是内部可达的，它的用途有 2 种：

- 普通的 PC 使用环回接口判断 TCP/IP 协议栈工作正常与否，TCP/IP 协议栈工作的正常直接影响应用工作正常，所以通常用于应用的测试；
- 网络设备如同交换机、路由器使用环回接口 IP 地址作为网络上的标识，因为接口始终是可达的，因此作为标识最为合适，可以长期判断网络设备的可达性。

环回接口只是众多虚拟接口中的一种，也是最常见最简单的一种，虚拟接口通常都为了网络层协议服务，所以在后面的章节中会有更多的介绍，这里先做一个基础的铺垫。

2.16. 最大传输单元 MTU 与路径最大传输单元 PMTU

在第 2.1 节中提到了要介绍 MTU，早在 2.10 节介绍 PPPoE 的时候借 MRU 提到了 MTU，那 MTU 和 MRU 具体是什么关系?

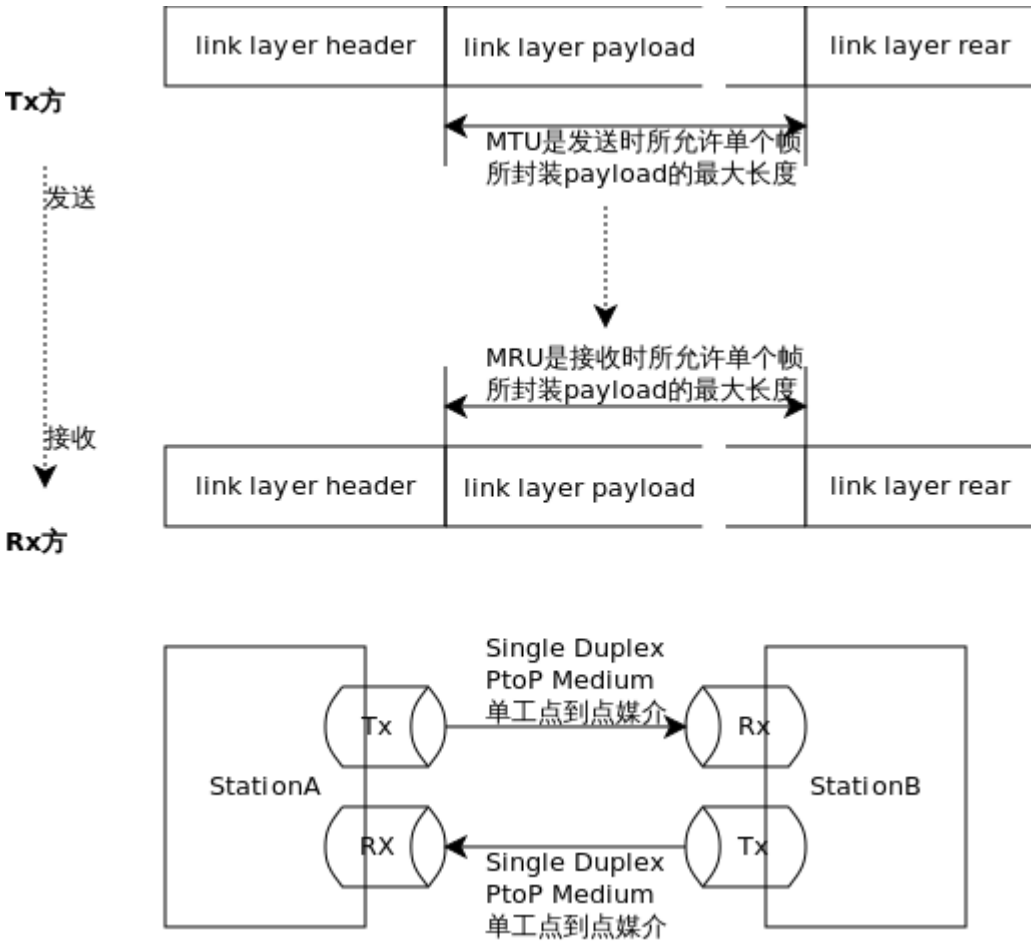


图 2-68 MTU 与 MRU

- MTU 叫最大传输单元，是链路层面向网络层的服务参数，单位字节，也就是说当网络层交给链路层的载荷 payload 超过 MTU 时，就提示网络层，这个 payload 太大了，不能一次发送完，需要多发几次，这种原本 1 个 payload 要分开发送在网络层叫分片 *fragmentation*，分片这种费时不讨好的操作可以由网络层完成，也可以由链路层完成，这取决于链路层是否提供这种服务（大部分链路层都不提供这种服务，也有网络层向链路层提要求的，比如 IPv6 就要求所有链路层提供 MTU 至少是 1280，如果实际链路 MTU 小于 1280，那么就请链路实现链路层分片吧），流行的 MTU 基本上是 1500 字节，如图 2-7 中 802.11 虽然可以支持超过 10 000 的 payload，但大部分 802.11 网卡提供给网络层的 MTU 也都是 1500，它所支持大 payload 是为了帧聚合，图 2-5 中 802.3 封装的以太网 MTU 则是 1492；
- MRU 叫最大接收单元，在同一个节点上，链路层芯片的 MTU ≤ MRU，单独列出来的目的是为了与对端协商，虽然大部分链路层如以太网和 WLAN 都是不协商的，但 PPP 是协商的，也就是希望对端节点的 MTU = 本端的 MRU，这样可以使双方的 MTU 保持一致，因为以太网比 PPP 更常见，所以 MRU 通常不为人所知。

它们的确很好记忆，MTU 用于 Tx 方，也就是链路的发送方，MRU 自然是用在 Rx 方，还是从介质的角度对比一下：

- 在共享型介质（半双工 half duplex 类型）上，所有介质上的节点都把 Tx 和 Rx 连接到相同的介质上，所以所有节点的 MTU = MRU，已经绝迹的同轴以太网和 Hub 以太网、时髦的 WLAN 都是这种类型，所以很好理解为什么以太网上很少提 MRU，以太网上提 MTU 通常是因为 802.1Q 或者更复杂的封装；
- 在点到点介质（如图 2-68 上半部）上，链路由 2 条单工（single duplex）介质组成（全双工 full duplex 其实就是 2 条单工组成），其中一条介质专门供 StationA 发送给 StationB 方向，另外一条正好相反，目前市面上的网络线缆基本上都是这么组成，在这种环境中 StationA 上的 MTU MRU，StationB 上也一样，但 StaionA 的 MTU 是否和 StationB 上的 MTU 相等呢？这就是需要协商，在 StationA → StationB 方向，StationA 将己方 MRU 发送给 StationB，StationB 就知道本端 MTU 该怎么调整了，反之亦然，在 MTU 可调的 PPP 中就是这么实现的，而以太网 MTU 固定就是 1500 或者 1492，所以即使交换机带来了点到点介质，依然无需修改或协商。

那么 MTU 在网络层传输过程中是什么样状况呢？那要从网络层传输的特点分析：

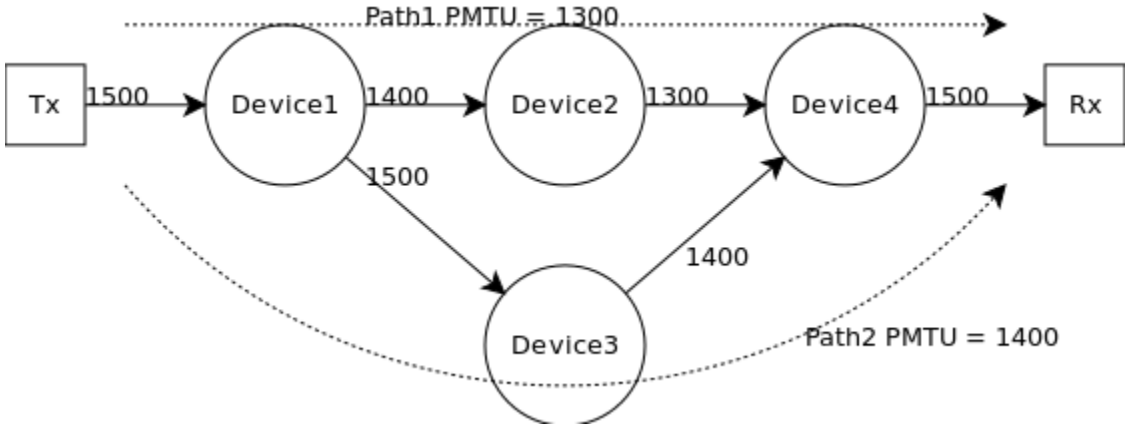


图 2-69 PMTU 在网络传输中的定义

在上图中假设 Tx 和 Rx 网络层之间由 4 台设备分别是 Device1~4 连接而成，Tx → Rx 方向可以选择两条路径 Path：

1. Tx → Device1 → Device2 → Device4 → Rx，每个箭头表示 2 者之间的链路，MTU 分别是 1500、1400、1300 和 1500，4 者中最小值为 1300；
2. Tx → Device1 → Device3 → Device4 → Rx，MTU 分别是 1500、1500、1400 和 1500，最小值是 1400；

两条路径各自对应自己的 PMTU，PMTU 的取值是取完整路径上最小的 MTU，所以 Path1 的 PMTU 是 1300，Path2 则是 1400，这也说明了即使 Tx 网络层交给链路层时即使尺寸足够，也无法保证能够毫发无损

地传递到 Rx，payload 超过 PMTU，还是要分片。如果还无法保证 Tx → Rx 始终选择相同的路径，那么 PMTU 还有可能随着路径而变化。为了避免这种分片情况的出线，大部分链路层都向最流行的以太网看齐，纷纷表示支持 1500 字节的 MTU，这也说明了以太网在目前链路层的强势地位。当然也不可能强求所有链路层都臣服于 1500 字节，所以网络层还是提供一些手段来探测 PMTU 值，通过实时探测结果修改网络层数据包的合理大小，避免分片，在网络层一章中会进行介绍。

2.17. 计算吞吐量

链路吞吐量（link throughput）也被称为线速（*line speed*），有时候也被叫做带宽 *bandwidth*，比如 1Gbps 以太网，它的速率是 1 000 000 000 bit/s，它的单位是位，而不是字节，记住在提到链路速率的时候，单位都是 bit 而非 byte，两者可是有 8 倍之别，另外 1Gbps 指的是物理介质上传输纯 2 进制 bit 的速率。可是很多人会说，网络下载的时候怎么看都是只有 10M 呢？下载速率和线速计算的差别在哪里？看下面这张图可能会有所了解：

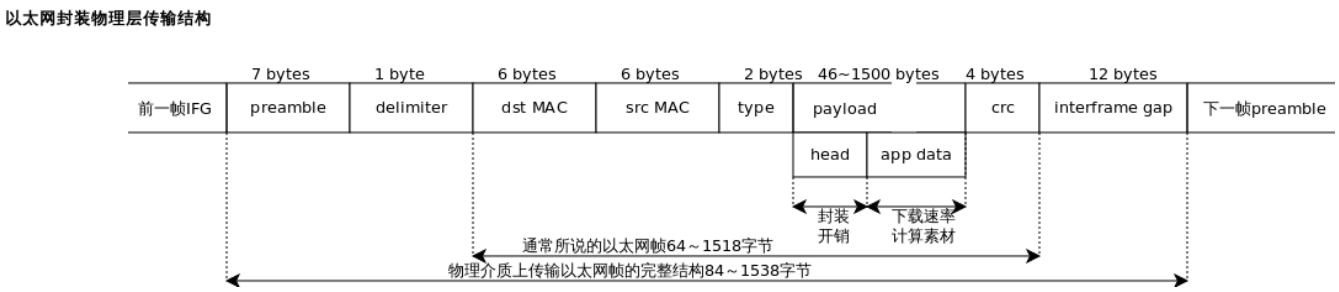


图 2-70 以太网线速计算与下载速率

大家对图 2-5 中以太网封装和 802.3 封装都比较了解了，那么是不是以太网就按照图 2-5 的结构在介质上传输呢？从图 2-70 来看，在介质上传输的时候，还有 20 个字节附加的东东：

1. 7 个字节的 preamble，前导码；
2. 1 个字节的 delimiter，定界符；
3. 在尾部还有至少 12 字节的 *IFG*（*Inter Frame Gap* 帧间隙），也就是每个帧在发送完 CRC 后就要发送至少 12 字节的 IFG。

显而易见，所谓线速就是以尽可能的速度在单位时间内传输数据帧，能想到的极致速度就是以背靠背（一帧紧接着一帧）方式传输数据帧，就以太网而言当 **IFG** 为 **12** 字节时就是背靠背了，再小就被认为是冲突，反而不是极致速度了，所以图 2-70 已经是前导码紧跟着前一个 IFG，IFG 后还紧跟着后一个前导码了，我们以最小帧长 84 字节计算一下 1Gbps 线速可以发送多少数据帧：

$$\frac{10^9}{(8\times 84)}=1488095.238095238$$

我们可以稍微整形一下就是 1 488 095.24 fps（frame per second），如果是按 1538 字节的最长帧计算就是

$\frac{10^9}{(8\times1538)}=81274.382314694$ ，整形后变成 81 274.38 fps。

这只是计算帧速，大家还是关心下载速率怎么计算的，下载速率是有应用程序 app 自己计算的，它计算的素材就是图 2-70 中的 app data 部分，单位是字节，以太网部分的封装开销 84 – 46 = 38 字节，还有网络层、传输层的封装开销（图 2-70 中 head 部分）是可变的，以常见的 IP-TCP 固定封装各 20 字节来计算，那就是 40 字节，所以常见应用数据在以太网传输时的开销是 38 + 40 = 78 字节，我们可以使用传输效率这个概念来计算下载速率：

$efficiency=\frac{data}{(frame\ length)}$ ， $frame\ length=data+$所有封装开销

故这个公式转变为

$efficiency=\frac{data}{(data+所有开销)}$

那么一个 84 字节长的数据帧，以 IP-TCP 封装算，真正的应用层 data 只有 6 字节，所有开销 78 字节，所以 efficiency = 6 / 84 = 7.14%，真是低得可怜；最长帧 1538 字节长，data 有 1460 字节，efficiency 计算出来是 94.93%。我们的下载速率最大值就可以这么计算了：

线速 * 7.14% ≤ 下载速率 ≤ 线速 * 94.93%

如果取线速为 1Gbps，那么下载速率就在[71428571.43bps, 949284785.44bps]这个区间内，单位 bps，要换成常见的 Bps（Byte per second），那么就是[8 928 571.43Bps, 118 660 598.18Bps]，也就是[8.9MBps, 118.7MBps]之间，这就是 TCP 应用在 1Gbps 以太网上的吞吐量，被称为应用吞吐量。通常 PC 的网卡是跑不了线速的，大部分网卡能够跑到线速的 1/10，所以我们的 PC 下载能够达到 10MBps 已经算是不错了（我的 PC 下载最多可以达到 25MBps），因为应用吞吐量还涉及操作系统以及应用程序，所以计算应用吞吐量并不能衡量一个网卡或者网络设备的性能。

在评价网络吞吐量时通常是按照 IP 包长的 efficiency 计算，这被称为 **IP** 吞吐量计算（单位是 bps），是业界测量网络设备性能的标杆，IP 包长的范围就是以太网 payload 的 46～1500 字节，所以其 efficiency 范围是[54.76%, 97.53%]，对应的 IP 吞吐量范围是[547.62Mbps, 975.29Mbps]。从图 2-11 来看，PPP 封装的 IP 效率可以达到 1500 / 1508 = 99.47%，很不错。从吞吐量的计算可以得出这么一些结论：

- 在开销固定的情况下，data 或者 payload 越大，吞吐量越高，但 payload 不能超过 MTU 值，所以一直强调 IP 要尽量达到 MTU，但不能越过（越过会引起分片，虽没有增加太多开销，但带来额外的损失），PMTU 探测也是为了在不带来额外损失情况下尽可能地提高应用数据吞吐量；

- 在 data 或者 payload 固定的情况下，开销越多，吞吐量越低，因此以太网封装要比 802.3 封装的吞吐量高，通常使用 802.1Q 封装后，吞吐量也会有所下降，但很多时候为了实现更多的功能，是需要占用一些开销的，也是值得的。

似乎一切都是那么顺理成章，但我们刚才的计算都是一个理论上的最棒值，实际网络中，一条链路上运行的并非只有单纯的应用数据，可能还有许多应用，许多其它的协议数据，甚至有许多错帧参杂其中，链路速率是被众多应用瓜分的。还有一个现实问题就是，刚才的计算都是在点到点介质上的计算，如果是在共享型介质计算吞吐量又当如何呢?

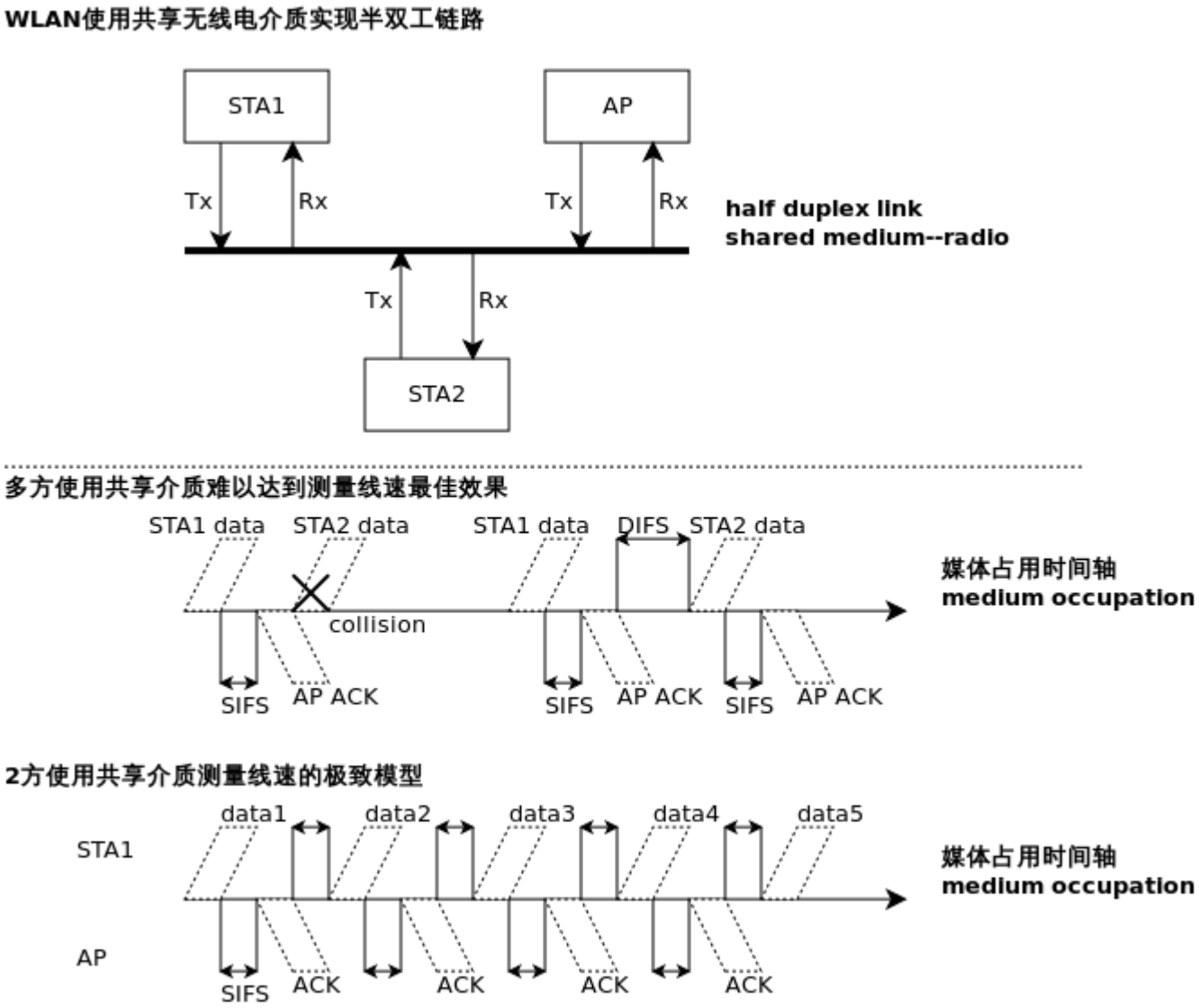


图 2-71 共享介质计算吞吐量

无线电和 Hub 一样是一个共享介质，从图 2-2 的分析，它必然是一个收发无法同时进行的半双工链路。在这种链路上测试吞吐量会有什么问题呢，从图 2-71 多方使用共享介质难以达到测量最佳效果来看，多方测试时容易产生冲突，或者需要等待时间较长的 DIFS，所以经常在号称 54Mbps 的 802.11g 网络中下载很难突破 1MBps。对于这种现状，测试 WLAN 吞吐量的方法就是只让 2 方共享一个无线介质，分别是 STA 和 AP：

- 找一个空旷、无线电干净无干扰的大场所，就 STA 和 AP 面对面的测试，这种测试对冲突的缓解是一定的，但冲突是一个可以测量、无法避免的因素，一个无干扰的场所很难存在，同时无线电还存在散射、多径、天线极向等因素，很难达到最理想的效果；
- 所以另外一个方法就是抛弃无线电，使用有线介质替代，因为天线终究是从有线介质获得数据，调制成无线电波后从天线发出去的，所以国外有的测试仪使用一种特制的金属介质將测试仪和 AP 的天线

端子连接，这种方法可以测试 AP 是否真正可以达到 54Mbps 的 802.11g 理论转发速率。

半双工链路和点到点全双工链路在计算吞吐量的区别在于半双工链路需要将接收方向流量也统计在内，也就是说 802.11g 带宽 54Mbps 指的是发送和接收合起来 54Mbps，而 100Mbps 全双工以太网则是发送 100Mbps 同时接收也可以 100Mbps。从图 2-7 来看 802.11 封装开销也要比以太网多，这种条件下 IP 在 802.11 环境下的 efficiency 也很难计算，目前测量比较好的结果大概可以达到 84%左右。而实际的情况呢，只能更糟，因为 WLAN 是一个不受管制的自由技术，所以干扰、冲突非常普遍，越普及则越严重，所以现在越来越时兴加强管理的运营级别 WLAN。不过还是搞 WLAN 的人那句话，能用有线以太网就尽量使用有线，实在是因为布线不方便或者手机这样没有以太网口的东西，才使用 WLAN，正确地看法应该是这 2 种技术是互补的。

2.18. 小结

链路层一章介绍了几种目前常见的链路层协议如以太网、PPP、PPPoE 及 802.11，以及它们的扩展协议如 802.1Q、802.1X 认证、802.11i 和 802.11e 等，每项协议都有明显的特点，要么适合互联，要么解决某些安全性问题、或者提高传输效率等等。在后面的章节介绍了环回接口、MTU 与 PMTU、链路吞吐量的计算方式、传输效率等方面的东西。

本章另外一个特点就是许多技术都是围绕着介质来讲的，介质和链路分离是 OSI（国际标准化组织 ISO 搞的一套架构叫 Open System Interconnection）的主意，在原生的 TCP/IP 中，把介质功能统一纳入到链路层了，链路层的作用很明显对网络层屏蔽了介质层面的不同，对 IP 层来说，链路层始终那么简单，殊不知，链路层技术才是当今计算机网络技术最风起云涌的阵营，风起云涌其实就在于 MAC、介质上大作文章，如以太网阵营就有运营商级别以太网 802.1ah、面向虚拟化则有 802.1qbg 和 802.1qbh、面向二层多路径有 802.1aq、面向管理的 802.1ag，介质上有 40G 以太网、100G 以太网，802.11n、802.11e 等等，都是当今网络界耀眼的明星。相比之下网络层和传输层的技术在 P2P 传输模式之后就鲜有革命性的架构了，这也是为何花了比较多篇幅来介绍介质相关的内容。围绕着介质，也可以对物理层设备、网桥、路由器做一个统一的定义：

- 物理层设备作用是延伸介质覆盖范围，功能是将不同介质上的物理信号调制到延伸介质上，如 WLAN 的天线和以太网的 Hub；
- 网桥覆盖了物理层设备的功能，它高级的地方在于通过实现链路层协议使物理隔离的介质之间相互通信——通过读懂数据帧的链路层结构，根据链路层结构信息将数据转发到不同的介质上，常见的网桥有以太网交换机（指常说的 2 层交换机）、WLAN AP，可以对比一下图 2-72 和图 2-73，可以发现 AP 将链路层帧封装从 802.11 换成 ethernet 帧封装，而 IP 以上部分丝毫未损，这是非常典型的网桥功能；

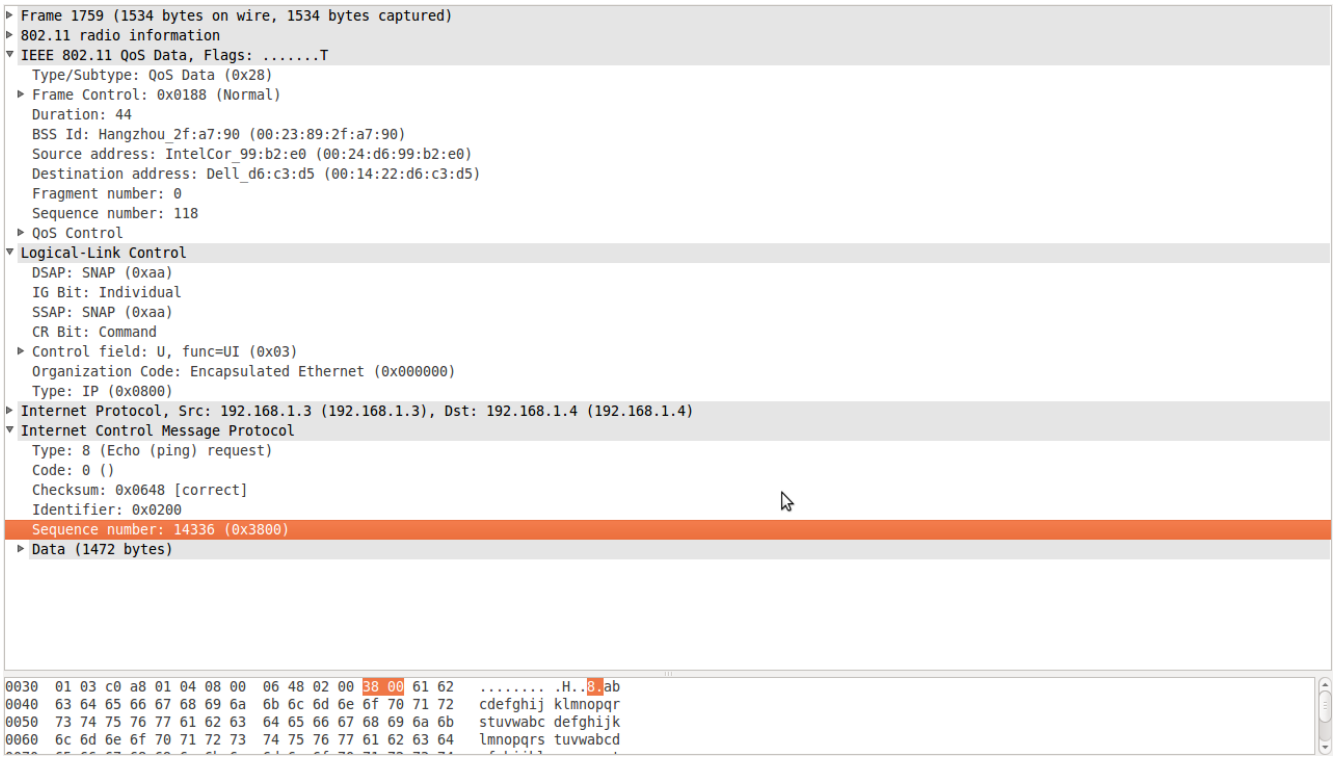


图 2-72 WLAN AP 从天线接收到的 802.11 帧

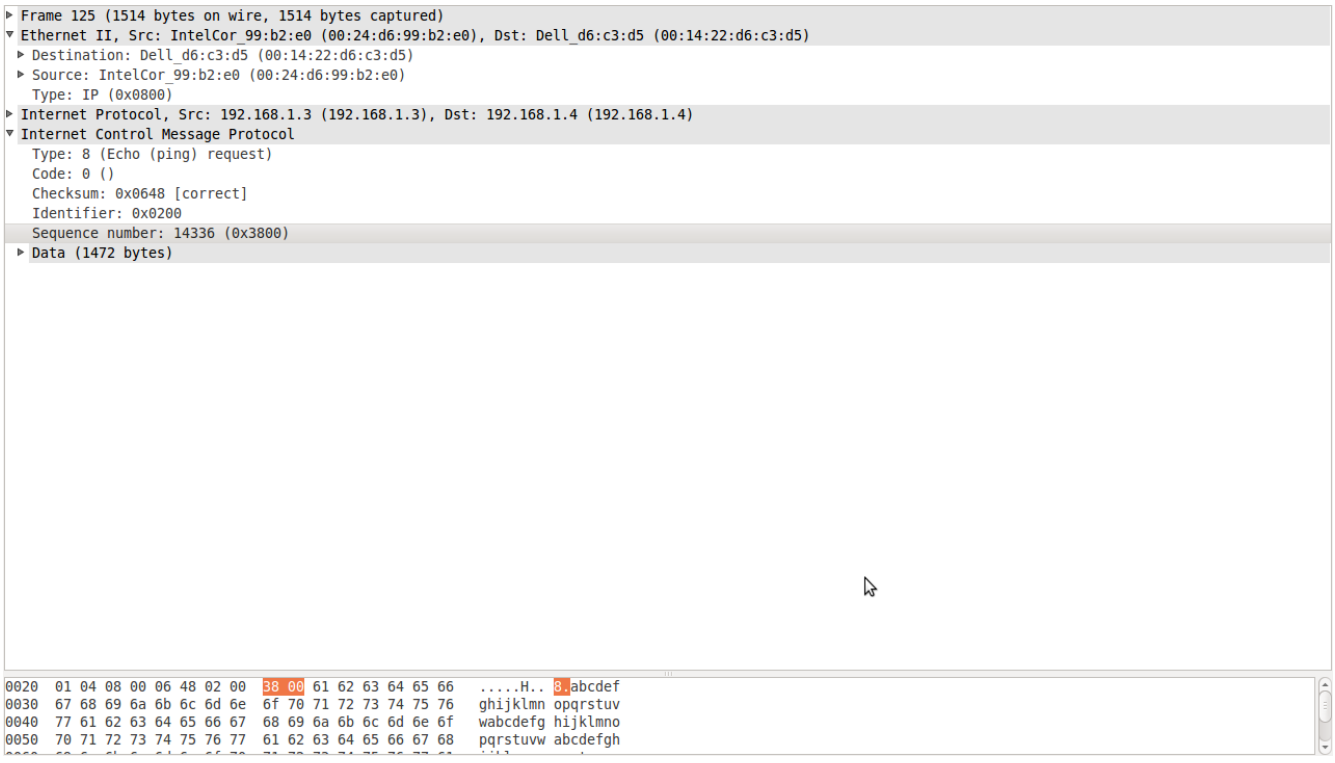


图 2-73 WLAN AP 从以太网接口发出的 ethernet 帧

- 路由器又覆盖了网桥的功能，它更高级的地方在于通过实现网络层协议使相互隔离的链路之间相互通信——通过读懂数据帧封装的网络层协议结构，根据网络层信息將数据转发到不同的链路上，常见的路由器有三层以太网交换机、防火墙以及路由器；

- 现在还有一些设备使用了隧道技术，比如链路层封装到网络层协议，链路层封装到链路层，对于这样比较复杂的设备通常就需要另外分析了，对于网络设备具体怎么分类对于现实已经不那么重要，市场始终关注的是能不能实现需要的功能、能不能有效地降低总拥有成本，这也就是为何会出来一个三层交换机，而不是二层路由器出来。

链路层如果要继续铺开讲的话，每一项技术都可以是一本几百页的大部头，更乐于以专题的方式来介绍，对于 TCP/IP 整个框架的理解而言， 并不需要太深入。

练习

2.1. 回想一下在一个房间里 2 个人交谈和 10 个人同时交谈的差别，嘈杂的环境和共享介质的冲突非常类似，从这理解以太网的 CSMA/CD 和 WLAN 的 CSMA/CA 的原理。

2.2. 將人的口和耳理解成发、收器官，2 个人面对面对话时是否一人说一人听有效，还是 2 人同时听说有效，从这里理解单工、半双工和全双工介质或者链路。

2.3. 使用您的 Wireshark 尝试抓取一下以太网接口收发的数据帧，检查一下是 ethernet 封装还是 802.3 封装。

2.4. 如果您使用了 WLAN，常识使用 Wireshark 抓取无线的数据帧，看一下是否能够收到 802.11 MAC 层数据帧，是不是 ethernet 封装的，想想怎么样去找纯正的 802.11 封装数据帧。

2.5. 是否配置过 PPPoE，如果您的 ADSL 宽带接入是需要在电脑的一个小程序输入用户名、密码的，多半是 PPPoE 拨号，用你的 Wireshark 把完整的拨号过程抓下来吧，看看本章介绍的 PPPoE 是否类似。

2.6. 使用过拨号 Modem 或者 3G 上网卡吗？它用的也是 PPP 拨号，但是 Wireshark 抓到的包却可能是 ethernet 封装的或者其它封装，还看不到任何 PPP 的痕迹，想想这是为什么？

2.7. 如果使用 Windows，在命令行窗口输入 ipconfig /all 看看是什么东东，仔细读懂每一行信息，不知道的英文单词记得查 google，如果像我一样使用 Linux，使用 ifconfig 吧。

2.8. 有遇到使用以太网需要输入用户名、密码之类信息的吗？如果是，有可能是遇到 Portal 或者 802.1X 了，能否使用 Wireshark 来亲自判断一下？

2.9. 想一下除了密码外还有什么方式保护您的身份信息比较稳妥？

2.10. 您的 WLAN 是否需要输入密码才能使用，如果是可以通过无线客户端工具检查 AP 是 WEP 还是 WPA/WPA2 吗？如果是 WPA/WPA2，请问它是个人级还是企业级？

2.11. 您是否拥有臭名昭著的 WLAN 蹭网卡，是否能够成功蹭网，对于 WEP 和 WPA/WPA2 哪种更容易蹭上，如果是企业级 WPA/WPA2 还能蹭上吗？如果 AP 是您自家的，请对比一下 AP 设置 MAC 过滤和无 MAC 过滤情况下蹭网卡是否可用？

2.12. 如果您找到一块可以抓去 802.11 MAC 帧的无线网卡，扫描一下共有几个 WLAN 信道工作，是否能够去侦听这些信道的数据，看看是否是加密后的数据帧，找一下是否有不加密的信道？

2.13. 使用 ipconfig 和 ifconfig 命令时是否检查到 loopback 接口，它和其它接口有何不同，它的地址是什么？对比 ping 127.0.0.1 和 ping 127.255.255.254 的结果，换成 ping6 ::1 和 ping6 ::2 呢？

2.14. 使用 ipconfig 和 ifconfig 检查每个接口的 MTU 是 1500 吗?

2.15. 计算吞吐量时为何网络层数据包大小越接近 MTU，但又不超过 MTU 效率最高？在上网的同时通过 Wireshark 抓包时留意一下 IP 包的大小是不是 1500。