互联网架构的设计哲学

V1.0

中国科学技术大学软件学院 2013 年网络程序设计课程全体同学 翻译

赵骞理 整理 孟宁 修订

说明

本文由 David D. Clark 在 1988 年写的论文 The Design Philosophy of the DARPA Internet Protocols 为蓝本编译整理而来，全文从今天（2013 年）的视角对原论文的内容进行了梳理，根据如今的互联网现状修订增删了部分内容，但由于水平有限难免对历史、对现状、对未来的理解有所偏颇，请读者谅解，也恳请读者将我们的偏颇疏漏之处邮件（mengning@ustc.edu.cn）告诉我们。

摘要

TCP/IP 互联网协议簇于 1973 年被首次提出，由 DARPA(The Defense Advanced Research Projects Agency)开发，已经被广泛应用于国防和经济领域各个层面。如今我们可以轻易找到大量研究论文和协议标准规格，它们详尽描述了 TCP/IP 协议簇的工作原理和具体规格。但是我们很难从这些论文和标准中弄清楚互联网为什么一步步地变成了今天的样子。比如 IP 协议是基于无连接数据报（Datagram）模式的，但其中详细的设计动机已经被人们大大地误解了。只有理解历史才能预测未来的发展趋势，这篇论文就是试图找出互联网协议演化至今的背后逻辑。

1.引言

DARPA 曾开发了一系列分组交换（Packet Switching）网络协议，其中就有 IP 和 TCP。现在这两个协议是美国国防部网络互连的标准，并被广泛应用于商业网络环境中。IP 和 TCP 的设计思想已影响了其他协议簇的设计和开发，尤其对 ISO 协议标准的制订产生了重要影响。

尽管互联网协议的具体规格已完全公开，但想要确定如此构思与设计的动机及权衡过程仍有些困难。

事实上，从最初的协议提案演化到如今的协议标准，它的设计理念已经有了非常大的变化。比如说，最初并没有特别强调数据报和无连接服务，但如今它们已成为了互联网架构的基础。另一个例子是将架构分层为 IP 层和 TCP 层，现在看来这是很基本的架构设计，但它并未出现在最初的协议提案中。这些互联网架构设计上的演变，都是在不断重复的实践过程中产生的，并最终形成了当前的互联网标准。

如今互联网架构依然在变化，比如经常会有新的协议扩展挑战已有的某个设计原则。因此，理解互联网架构设计演化的历史是非常有意义的，它使我们在面对互联网架构上的变化时，能够清晰地理解该变化历史上的来龙去脉，以便做出更明智的决策。比如在 ISO 协议标准化工作中经常可以看到 TCP/IP 协议簇的影子。因此，深入理解互联网架构的设计哲学对于从事协议设计和标准化工作的人来讲是有益的。

本文探讨了互联网架构设计的最初目标，并讨论了这些目标和协议中重要特征之间的关系，也对当今互联网架构的变化趋势有少许延伸。

2.互联网架构设计的核心目标

互联网架构设计的核心目标是开发一种能够将现存不同类型的网络互连起来充分利用（复用）的有效技术。互联网出现之前各种不同类型的网络之间是很难进行通信的，下面的详细解释会将有助于进一步弄清这个目标的意义。

互联网是由网络互连起来组成的，这样互连起来的网络可以提供更大范围的服务。最初是为了把 ARPANET 网络和 ARPA 分组无线电网络（packet radio network）连接在一起，为了让分组无线电网络的用户能访问 ARPANET 网络上的大型服务器。尽管当时局域网（LAN）还没有出现，但是设计人员就已经假设将来会有其他类型的网络要连接进来。

为了实现网络互联的目的，一种可选的方法是设计一个统一的系统把现存的网络连接进来，该系统可以使用多种不同的传输介质，也就是多种传输介质网络。虽然这样高度集成是有代价的，但性能会很好。可是另一方面如果想让这样的互联网系统在实际应用中成功的话，就必须合并现存的不同网络架构，此外网络还有管理域的问题。因此，要实现这样一种方式的互联网似乎有些过于雄心壮志了。

复用技术的选择就是分组交换。电路交换（Circuit Switchinig）的方法也被考虑过，但网络所要支持的很多应用是基于分组交换方式的，比如远程登录，需要互连进来的网络也是分组交换网络。所以分组交换技术成为了互联网架构的基础。

核心目标的最后一个方面是选择网络互连的具体技术。由于存储转发（store and forward）分组交换技术在 APRANET 网络中被充分研究并获得了成功，所以利用互联网的分组交换作为一个关键点让网络互联起来，也就是我们所说的网关（Gateway）的核心功能。

所有这些设计决策构成了互联网架构的核心：将不同类型的网络利用网关互联起来实现分组交换方式的通信，其中网关是基于存储转发算法实现分组传输（Packet Forwarding）的。

3.互联网架构设计的具体目标

前面已经提过在网络架构设计中的核心目标，那就是“effective（有效）”，但是却没有说明有效互联具体要达成什么样的目标。下面列表中设定了网络架构设计的详细目标：

1) 面临故障时的通信生存能力；
2) 支持多种类型的通信服务；
3) 支持不同类型网络的接入；
4) 支持资源的分布式管理；
5) 要考虑成本效益因素；
6) 主机接入互联网的代价要低；
7) 支持网络资源的计费统计。

这些具体目标看起来是网络系统需求的一个完整列表。可重要的是我们需要理解这些目标先后顺序的重要性。如果顺序发生改变，将会导致完全不同的网络架构。例如，对于军用网络，意味着在传输数据时可能遭到敌方破坏的复杂情况，那么通信生存能力就会成为首要目标，而计费管理能力则是排在最后的目标。在战争期间，我们更关心如何尽快地将收集到的信息可靠地传递到目的地，而不考虑通信费用。所以最初大家并不关心资源的计费管理，而对于商用网络架构的设计，资源的计费管理应该放在首要位置。

同理，成本效益因素也是我们的目标，但是排在资源的分布式管理和支持不同类型网络接入之后。一些流行的商用网络会专门针对特定的传输介质做优化，比如基于长途电话线实现的存储转发网络可以在很低的成本下通信良好，但对于其他类型网络的支持可能却很差，因为商用网络架构设计会更多考虑成本效益问题。

如果您仔细考虑了上面的目标清单，会发现它是互联网架构设计决策的优先次序。接下来几部分将讨论这个清单和互联网架构特征之间的关系。

4.面临故障时的通信生存能力

互联网最重要的目标是能够持续地提供通信服务，即使在部分网络和网关出现故障的情况下。具体来说，如果两个实体通过互联网进行通信的过程中出现故障使得通信临时中断，它能够重新配置网络恢复通信服务，然后实体间的通信可以继续，而不用重新建立更高层的会话。更具体地说，传输层协议并不总向客户端报告网络临时中断故障，它总是假定故障可以很快恢复，反复尝试后确定网络无法恢复时才向客户端报告。换句话说，在传输层之上只有网络完全不通时才会出现通信失败。架构完全掩盖掉了任何短暂性的错误。

为了实现这个目标，通信会话的状态必须被保存。典型的状态信息包括已发送的数据包数、已确认的数据包数，还有流量控制信息等。如果丢失了这些状态信息的话，底层协议就搞不清楚数据是否丢失了，应用层不得不进行同步。如果不容许丢失这些重要状态信息，就意味着状态信息必须要保存好防止丢失。

有些网络架构是把这些状态信息保存在网络中间的分组交换节点上。在这种情况下，为了使信息不丢失，就需要把这些状态信息在传输节点之间复制。在分布式状态下管理这些状

态信息的健壮算法很难实现，所以这种分布式的网络几乎都不提供应付失败的机制。而互联网架构选择的方案是把状态信息收集起来保存在网络通信的两端。这种方式称为“fate-sharing（生死与共）”。它的意思是通信的一端消失了，那么另一端也同时将自己保存的状态信息丢掉。具体来说，传输层的状态信息保存在主机中，主机连接到整个网络并且使用整个网络上的服务。

“fate-sharing”模式比“复制”模式有两大优势：第一，前者能够不受任何网络中间节点故障的影响，而后者却不能；第二，前者工程实现更容易。

“fate-sharing”模式意味着两点：第一，中间分组交换节点或网关都不保存当前通信会话的状态信息，换句话说，它们是无状态的分组交换。因此这种网络架构常常被称为数据报网络。第二，在这种架构中主机的作用更重要，因为它不像有些网络本身提供可靠的数据传输服务，而它利用主机端的传输层算法保证了数据被有序传输、丢包重传和确认等，而主机上的应用程序不必关心数据传输上的细节问题。

尽管面临故障的生存能力在互联网架构设计中是首要目标，但是相对于将现存网络互连起来这一核心目标而言毕竟是第二位的。因为单一的多种传输介质网络架构或许会有更强的生存能力，但它不利于将现存网络互连起来这一核心目标。

5.支持多种类型的通信服务

互联网架构的第二个具体目标是可以支持多种类型的通信服务。不同类型的通信服务在传输速度、延迟、可靠性等方面差别很大。传输服务中最传统的类型是一种双向可靠的数据传输服务。这种服务又被称为“虚电路”服务，适用于远程登陆和文件传输这类应用，这是在使用 TCP 的互联网架构中首要提供的服务。很早人们就认识到即使是传统的“虚电路”服务也无法满足多样性的需求。比如其中的远程登陆和文件传输，前者要求服务延迟不能太大，但对带宽并没有过高要求；而对于文件传输，更关心的是数据的吞吐量而不是传输延时。此前，TCP 尝试同时提供这两种服务，结果并不理想。

最初 TCP 的设计目标是让它足够通用，可以提供任何需要的服务类型。但是，等弄清楚不同服务类型的特点后，发现使用一个协议来承载不太可能。

XNET(cross-Internet debugger)是不适用 TCP 服务的例子之一。原因如下：首先，一个调试协议不要求可靠性。这个观点看似有点奇怪，但试想一下只有在高负荷或存在大量错误的环境下才需要调试器，在这样的条件下要求可靠传输那不是自相矛盾了（调试器的目的是找到网络中的错误，却又要求网络无错误的传输数据）。所以更好的做法是，建立一种服务，能够处理每个通过网络的数据，而不必要求每个字节都可靠地按序交付。其次，假如 TCP 真的足够通用有能力处理各种各样的客户需求，那它必然会相当复杂，问题是指望在调试环境下支持这种复杂性本身就不太现实，事实上在调试环境下连操作系统中一些基本的服务都可能没有，比如计时器。所以最终 XNET 被设计成为直接运行在最基础的数据报服务上。

另一个 TCP 不适用的服务是实时通信传输，电话会议和电视直播就用到了这一服务。实时通信传输服务的首要要求是，在传输数据包时，有更小和更平滑的延时，而并不要求很高的可靠性。也就是说在电话会议中传输过来的语音及时且连续就行，不需要每个音节都很清晰准确。大致过程是应用层将模拟语音数字化，并打包这些二进制数据，然后按序通过网络发送。这些数据包必须有序才能保证还原成正确的语音信号。如果一些数据包未按预期到达，那就不能及时的再现语音了。在使用 TCP 的情况下做了一系列的研究对传输延时进行控制，最后惊讶地发现，传输的可靠性才是导致延时的主要原因。这是因为典型的可靠传输协议发现丢包时会发送一个重传请求，并会推迟后续数据包的传输，直到丢包重传成功了才会滑动

发送窗口传输后续数据包。这种情况一旦发生会造成数倍的额外延时，也可能会完全中断。导致最终得到的语音数据是完整的，但却只能听到断断续续无法识别的声音。这显然是不合适的，事实上，对于丢包的处理可以非常简单。那就是当遇到丢包时使用一段空语音（没声音）替代，因为在大部分情况下这对听者分辨语音内容没有影响，然后继续传输后续数据。即使产生影响了，要么可以使用高级纠错，要么听者让对方重复一遍就行了。这样似乎问题解决了，但对于像电视直播这种需要较大网络带宽的实时通信传输服务，互联网所能提供的服务质量依然差强人意，因为基于无连接数据报的互联网架构无法保证在通信的两端之间提供一个持续且稳定的数据流。尽管多年来我们探索了在下文提及的优先传输队列方法，以及资源预留协议 RSVP 等服务质量 QoS 技术，但问题并没有得到彻底解决，或许将来基于“流”的抽象封装替代目前基于数据报的封装层，能给此问题提供理想的解决方案。

在互联网架构建立之初就有满足传输服务多样性的目标，对于诸如可靠性、延时、带宽等等要求各不相同的服务，必须可以兼容它们同时传输。

为了实现这个目标，设计者们将 TCP 和 IP 从原来架构中的单个协议扩展成现在的两层。也就是传输层和网络层。传输层的 TCP 提供一类特殊的服务：支持可靠有序的数据流传输，而网络层的 IP 则试图提供一个基础，与多种类型的服务相隔离。而这个基础就是 IP 数据报，在 IP 数据报之上支持持续连接来传递报文，但是单纯使用数据报传输，其可靠性得不到保证。我们需要建立独立于数据报的可靠传输服务（通过在高层实现确认重传机制），或者使用底层网络原有的可靠性服务。此外还有一种与 TCP 对应的协议：用户数据报协议 UDP，不同于 TCP，它面向无连接且不可靠的。对于互联网中基本的数据报服务，它可以提供一个应用层的接口。

互联网架构中对于多种服务的支持不应该由下层网络来提供，否则会背离互联网架构设计的目标。因而，我们希望由主机和网关来提供算法，将各种服务从基本的数据报服务（IP 数据报）中抽象出来。比如有这样一种算法：选出有延时要求但是不要求可靠服务的数据报，将它们放在传输队列的前段，对于这类数据报，生命周期一旦结束就将其丢弃。这样就可以保证它们尽量快的传输，以满足延时要求，这就是通过优先传输队列的方法提供实时通信传输服务。此外，那些要求可靠性的数据报就放在队列后段，对于这些数据报，不管它们在网络中存有多久都不会被丢弃，以保证最终能正确传输，满足可靠性。

在没有下层网络的支持下提供多种服务比最初的设想要难得多。其中最大的问题在于，针对单一服务设计的网络，在支持其他服务的时候会有各种问题。一般说来，不管是否要求可靠性，一个网络在设计时，都默认提供可靠的服务。并且会把延迟作为可靠性的一部分。比如 X.25 定义的接口是面向可靠传输的，并且无法去掉这一特性。因此，虽然互联网在 X.25 上成功运行了，但它并不能提供其他类别的服务。也有例外，有些网络有内部数据报服务，对于它们允许的服务可以提供好的支持，但这类网络毕竟很少，尤其是在远距离通信环境下。

6.网络的多样性

互联网架构获得成功的重要原因在于它能够整合不同技术的网络架构，包括军事和商业领域的网络技术。互联网架构非常成功地达成了这一目标：它工作在多种网络之上，包括长途网（ARPANET 和 X.25 网络等）、局域网（以太网、环网等）、卫星广播网（DARPA 大西洋卫星网传输速率为 64K b/s，DARPA 实验宽带网在美国国内传输速率为 3M b/s）、无线电分组交换网络（DARPA 无线电分组交换网络以及英国一个实验的无线电分组交换网络和由业余无线电操作人员开发的网络）、多种多样的串行连接（从 1200b/s 的异步连接到 T1 网络）以及

各种其他设备，包括计算机内部总线和通过其他网络套件提供的服务（如 IBM 的 HASP 提供的更高层的传输服务）。

网络架构通过对网络提供的功能做出最低限度的假设来达到这样的适应性。基本假设是：网络可以传输数据包或数据报，数据包必须是合适的大小，可能最小是 100 比特，并且应该具有合理的但不完美的可靠传输。如果不是点对点连接，则网络必须具有合适的寻址方式。

有很多服务是明确不应该由下层网络提供的，包括可靠性或按序传递、网络级广播或组播、传输数据包的优先级排序、对多种服务类型的支持、还有丢包或延时等具体细节。如果这些需求被要求了，为了适应互联网中的不同网络，那就需要根据网络本身是否支持这些服务选择直接使用或用网络接口软件在网络终端提供额外增强功能来模拟这些服务。这看起来并不是好的做法，因为这些服务都必须在多种不同网络及其连向网络的主机上重新设计实现。因此在传输层设计这些服务将更简单，例如通过 TCP 实现的可靠传递，仅需设计一次，可以运行在差不多所有的主机上。这样，为一个新网络移植这些协议软件通常非常简单。

7.其他的具体目标

前面三个具体目标对互联网架构设计的影响最为深刻，其余的具体目标重要性低一些，在互联网架构设计中或有所忽略或设计的并不完善。支持资源的分布式管理这一目标在某些方面做的很好，比如互联网中有很多网关，它们并不是由同一个机构管理，而不同机构会有不同的路由算法，这时就必须要求它们能够正确地交换路由表，不管他们之间是否相互信任。事实上在各个网关中存在着各种各样的算法，因为网关和该网关连接的网络常常是由不同的组织管理的。

曾经互联网在涉及到分布式管理时缺乏足够的工具，特别是路由问题。大型网络的路由决策受到计费管理等资源使用政策的限制，当然在少数情况下我们可以手动配置路由表的方式解决，但这很容易出错，且不够灵活。随着网络规模的扩大，各种各样的网络设备逐渐增多，对原本成功的资源分布式管理方式带来了更大的挑战，缺乏足够的工具来分布式管理网络的问题更加突出，在解决这个问题的过程中也形成了诸如 SNMP 协议的标准用来支持集中化的管理工具，这在一定程度上缓解了网络管理问题。另一方面，互联网承载的应用和服务越来越多，这也给网络中间设备赋予了更多的职责，造成网络协议越来越臃肿复杂且设备成本越来越高，网络数据流量分布式决策的架构方式成了这些问题的根源。在单一组织管理下的网络中，将网络中的数据面（数据转发）和控制面（转发决策）相分离，从而将网络中所有网络设备的控制功能和管理功能集中起来由单一设备的控制器（逻辑上单一内部可以使用分布式）负责，这样就可以既解决了资源的分布式管理难题又解决了网络协议臃肿复杂造成的网络设备成本的提高和性能方面的损失。这种数据面和控制面分离的方法即是软件定义网络 SDN 的核心思想。互联网架构在未来几年最重要的变化可能是一个新的资源管理平台——网络操作系统，也就是 SDN 网络的控制器。但 SDN 并不能解决不同的管理部门间协作管理共用的网络资源问题，也就是说，网络资源的分布式管理依然是互联网架构中非常重要的一部分，而 SDN 局部地挑战了互联网架构的分布式原则。

很显然互联网架构并不完美，但该架构能够合理有效地利用网络资源。网络包的头部都相当长（一个典型的 IPv4 头部是 40 字节），如果要发送的数据包非常小，这种开销是显而易见的。更糟的情况是基于单个字符的远程登录数据包，携带 40 个字节的头和一个字节的数据。这对任何协议的设计来讲都是非常困难的，只能进行合理的效率优化。在另一个极端，大文件传输 1000 字节的数据形成的 IP 数据包，头部的开销只有百分之四。

效率低下的另一个来源是丢包重传。由于互联网架构不支持丢失的数据包在网络层面上恢复，那就需要重新传输丢失的数据包，完整地从源端重新传输到目的端。这意味着，数据包可能多次跨越多个中间网络，而在网络层面的复苏不会产生这种重复流量。这是前面提到的由主机负责可靠传输服务的方式所带来的性能损失，但这样网络接口代码要简单得多。如果重传率足够低（例如 1%）则增加的成本是可以接受的。显然在 1%的丢包率状况下这样相当合理，但如果有 10%的丢包率，那在网络层支持可靠传输服务就是必需的了。

这样主机接入互联网的代价也许比其他方案成本要高，因为网络运行的一些服务必须在主机上实现，如确认和重传策略，而不是在网络中实现。最初在主机上实现这些协议有点令人望而生畏，其实随着经验的积累，我们可以将传输协议抽象出来，而不用区分不同类型的主机。这样在新类型的主机上实现这些协议也变得相对轻松，到今天这些协议可以运行在各种各样的设备上，包括个人电脑和计算资源非常有限的嵌入式设备。

主机负责可靠传输服务，如确认和重传策略等，当通信的另一端失去连接时这些算法依然在主机上驻留并继续进行确认或重传的动作，这就会造成网络和主机性能下降甚至瘫痪。这个问题是不可容忍的。因为最初的实验只涉及有限数量的几台主机，算法在主机上驻留造成的问题是可控的。然而，随着互联网的使用量大幅增长，这个问题就会时常浮现出来。为了让网络更加稳健，导致了“fate-sharing”方法产生，使得主机驻留算法产生的损失稳定在可容忍的范围内。

最后的具体目标是支持网络资源的计费统计。事实上，最初的互联网架构并没有考虑计费管理的问题，随着网络的应用范围从军事用途扩大到商业应用时，网络资源的计费统计能力在架构上的缺陷和缺乏监测工具等问题才逐渐得到研究。目前已经有大量的流量监测和计费统计管理工具，但流量统计的精确性问题依然会时常浮现出来，这是 IP 网络在架构上的固有缺陷，因为网络中间设备很难判断哪些数据包是由丢包重传引起的重复数据包，而这些重复传输的数据流量是由网络自身问题造成的，不应该由用户为此买单。

8.架构与实现

如前文所述，互联网架构需要提供相当大的灵活性，比如可以提供不同类型的服务、兼容不同类型的底层网络技术等。换句话说，这种架构要尽力避免限制互联网所能提供的服务范围。这意味着想要理解互联网提供的某种特定服务，我们不应该关注它的 IP 数据报基础架构，而是要关注主机与网关中 IP 层之上的软件和与此服务相关的下层网络技术。互联网的实现是按照互联网架构方式将网络、网关和主机连接起来构成。互联网的实现可以通过它所能提供的服务等级来区分，可以基于 1200b/s 的电话线路实现，也可以基于速度大于 1Mb/s 的网络实现。显然对于不同的具体实现有不同量级的数据吞吐能力。同样某些实现仅有数毫秒的通信延时，而另一些则有高达数秒的延时。显然实时语音这类应用在这两种实现上的运行效果完全不同。某些互联网被设计成有许多网关和通路的冗余，这些互联网的通信生存能力很强，因为出故障后，冗余的资源可以被重新配置使用。另一些互联网的具体实现中为了节约成本，将一些节点作为联通整个网络的唯一桥梁，那么一旦出现故障就会将互联网一分为二。

互联网架构有这么多种的具体实现，每种具体实现都需要设计者做大量工作，这成为架构实现中的难题。如何为具体实现的设计者提供一个指南，这个指南应能将具体实现的设计与能够提供的服务类型联系起来。例如一个设计者必须回答如下问题：如果整个服务需要提供稳定的数据吞吐能力，那么下层网络至少要拥有多大的带宽？对于具体实现中可能发生故障的某种情形，应将哪类冗余设计加入具体实现中？

大多已知的网络架构设计方案看来对回答这些问题都没有什么帮助。例如协议校验器有助于确认协议符合规格，但是这些工具无法解决服务类型属性所要求的性能问题，而只能严格地验证协议规格的逻辑正确性。当然验证逻辑正确性的工具无论在设计阶段还是实现阶段都是有用的，但是它们无法为性能相关的问题提供帮助。一个典型的经验是，逻辑正确性被验证后，却发现这种设计导致性能下降了一个数量级，类似的设计缺陷也时常被发现。对于这样个问题的研究结论是：困难不在协议内部，而在协议运行的操作系统环境中。在如此情况下也就很难从架构上解决这些问题，然而，向实现者提供一些指南确实是必要的。

另一种设计工具是模拟器，对于某种具体实现，模拟器可以检验在各种负载情况下能够提供的服务等级。目前已经有多种网络模拟仿真工具，比如 OPNET、NS2 和 Mininet 等。它们不仅可以在设计阶段验证协议的正确性，也可以提供包括路由器、交换机、服务器等网络设备模型，更重要的是它具有丰富的数据统计收集和分析功能，可以直接收集常用的各个网络层次的性能统计数据，能够方便地评估各种负载情况对性能的影响。此前大多数互联网实现的性能分析都是粗略的估算，但由于实际网络环境要比模拟的环境更加复杂，各种影响因素众多，因此互联网最初的设计者通常不大关心最后 5%的性能，相比之下他们更关心如何使用现有的资源来完成特定类型的服务，但之后会为了更好的成本效益不断做性能优化。

架构与性能之间的关系是一个很有挑战性的问题。互联网架构的设计者强烈地认识到仅仅重视逻辑正确性而忽视性能是一个重大错误。但互联网架构限制了性能，主要原因有两点：首先架构的目标不是保证性能，而是提供多样性；其次（也许更基本）最初并无有效的方法来描述性能。这个问题非常严重，因为互联网设计的目标是作为军方标准，而政府合同不能指望承包者超出采购标准。如果互联网架构设计考虑性能问题，那么性能参数必须要强制写入采购规格中，而制定有性能参数作为约束条件的采购规格是困难琐碎的，比如指明必须有能力每秒传输 1000 个数据包。所以这种性能约束没能成为架构的一部分。因此，只有到了最终验收的时候才意识到这些性能约束必须要加入规格之中，并且应该详细地说明，这样最终才能提供所要求的服务。对于这些制订采购规格的人，我们当时无法提供架构方面的指导。如今互联网已经融入了各行各业，架构上也越来越成熟，对采购招标中的性能参数我们的理解也逐渐深入，但在研究上架构的变化对性能产生的影响依然是一个很有挑战性的问题。

9.数据报

互联网基础架构特点是利用数据报作为底层网络中传输的实体。正如本文所述，数据报在互联网基础架构中非常重要，这有很多原因：首先，数据报使中间结点无需关心通信连接的状态，这意味着互联网在出错之后不用考虑状态就能直接恢复；其次，数据报提供了一个基础封装层，在此之上可以实现多种类型的服务，与只能提供固定服务类型的虚电路相反，数据报提供一个更基本的服务，在它之上可以建立其他类型的服务；第三，数据报代表了最小的网络服务假设，它允许多种类型的网络纳入到互联网架构实现中。使用数据报的决定是非常成功的，它使互联网非常成功地实现了它最重要的目标。

一个有关背后原因的常见误解：使用数据报是为了支持高层服务，而这些高层服务本质上与数据报服务相当。换句话说，人们有时认为传输层服务的应用需求是数据报服务，所以才使用数据报。事实上，这是比较少见的情况，虽然一些互联网应用使用不可靠的数据报服务，如时间服务器或 DNS 服务器，但大部分互联网应用更倾向使用比数据报复杂的传输方式，比如某些服务注重更强的可靠性、某些更关注延时，但几乎所有服务都比数据报更复杂。理解数据报作为一个基础封装层，而不是一种特定服务本身是很重要的。

10.TCP

在开发 TCP 的过程中出现了几个有趣且有争议的设计决策。TCP 在成为标准之前，曾经有几个主要的版本。如滑动窗口机制和端口地址结构的设计决策，在发布的 TCP 协议手册中讨论了它们，现在再讨论当时做出这些决策的原因似乎已经毫无意义，但如果想总结一下形成 TCP 的早期原因，那么讨论 TCP 的这些决策就很重要。完整的回顾 TCP 的历史需要过多的篇幅，这可能需要一篇与本文长度类似的文章，因此这里仅简要总结一下。

原始的 ARPANET 网络协议提供了两种流控制方式：基于字节和基于数据包。这样过于复杂，TCP 的设计者认为仅一种形式的管理就足够了。他们选择了基于字节的传输控制方式，而不是基于数据包。这是 TCP 基于字节数而不是基于数据包数进行发送和确认的原因。

这个决定的产生有多重因素，其中一些已无足轻重，而另一些则变得比原来还要重要。使用字节的一个原因是，想允许将控制信息插入到字节流的序列空间中，这样控制信息与数据能够同时被认知。为了支持 ad hoc（点对点）技术处理控制信息，将控制信息插入到字节序列空间的方法被废除了，但是这个方法已经被普遍应用，于是造成了实际应用中的复杂情况。

选择字节的第二个原因是，想要允许 TCP 数据包在必要时被分割成更小的包，来适应包尺寸更小的网络，但是随着 IP 从 TCP 中分离出来，这个功能被移到了 IP 层，由 IP 分片机制负责分割数据包。

选择字节而不是数据包的第三个原因是，如果必须重传数据，那么在发送端的主机中将重传数据与后续数据合并为一个大数据包更方便。当时尚不清楚这样做是否重要，如今证明这样大大提高了重传时的通信效率。像 UNIX 那样的系统有一个基于单字符的内部通讯模型，这个模型中的数据交互会经常发送很多仅包含 1 个字节数据的数据包。（也许从网络通信的角度来看，有人会认为这个做法很愚蠢，但对于交互式远程登录，确实如此，而且是必须的。）这种主机，常常被观察到产生数据包的洪流，这些数据包仅仅包含 1 字节的数据，这股洪流的到达速度远远超出一个缓慢的主机的处理速度(两个包到达主机的平均间隔时间小于主机平均处理一个包的时间)，结果就是丢包重传。显然重传数据与后续数据合并为一个大数据包在这种情况下就显得非常关键。

如果重传的数据仍是原先的未聚集的小型数据包，同样的问题会在每次重传过程中出现，这会对性能有无法忍受的冲击，所以这种运作方式不可接受。但是如果那些小数据包被归并入一个大的数据包，并进行重传，那么重传将会有效得多，实际上运作效果也很好。

另一个方面，既然基于字节的流控制方式会造成前述的这些问题，那如果流控制是基于数据包的，而不是字节的，比如主机在数据积累到一定大小之后才发送一个数据包，那么可能永远不会出现这种数据包洪流的问题。然而基于数据包的流控制按小数据包发送会严重限制吞吐量。如果接收方主机仅仅指定数据包的数目，而不管每个包中的字节数，那么它实际接收的数据量变化范围可达 1000 倍，这取决于发送方主机向一个数据包中装入 1 字节还是 1000 字节。另外传输过程中经过不同类型的网络时还可能会将数据包分割成更小的多个数据包，这会造成源端和目的端数据包个数统计上的混乱。

回顾过去，如果让 TCP 提供多种类型的服务，那可能必须既包括基于数据包的，也包括基于字节的流控制方式，和最初的 ARPANET 协议一样。

另一个与字节流相关的设计问题是结尾标志位 EOL，现在已经被 PSH 标志位取代，从协议中消失了。EOL 的最初构想是将字节流分割成若干个记录，将每个记录中的数据放入单个数据包中，这与丢包重传时合并数据包的想法不匹配。所以 EOL 的语义被弱化，表示流中数据的阶段性结尾，是一个或多个完整的应用层单元的结尾，它们会引起 TCP 或者网络中缓存

的一次冲刷（flush）。之所以是“一个或多个”而不是“就一个”，是因为可能会合并几个单元一起发送，来实现通过重组压缩数据的目标。但是较弱的语义意味着具体应用不得不建立特殊的机制，以在数据流中清楚地表示出各记录之间的边界。

在 EOL 语义的演化过程中，曾有一个不为人知的中间样式，它引发了激烈的争论。由于主机基于 TCP 的字节流模型的缓冲策略有时可能导致很大的问题，在主机上接收到的数据被装入一组固定大小的缓存中，每个缓存在被装满或收到 EOL 时返回给用户，可是数据包的到达是无序的，数据包内的 EOL 可能提前到达；更进一步考虑，在接收这些无序的数据包后，一个包含 EOL 的数据包导致这个仅仅装载了一部分的缓存被返回给用户，这种操作顺序导致下一个缓存中无序的数据被放入错误的位置，因为上一个缓存中部分空字节返回给了用户。

应对这些问题的建议是，用 EOL 填充数据包的尾部，使得整个数据包的大小为缓存容量的整数倍。换句话说，EOL 应该是将字节流映射为主机缓存管理的工具。这个想法在当时未被很好接受，因为用 EOL 填充数据包尾部看起来太特别了，而且错误并不经常出现。现在看来把字节序列空间与缓存管理相关起来的算法纳入 TCP 中是合适的。当时，设计者缺少理解在普遍情况下 TCP 应该做什么的洞察力。如今 TCP 经过长期的演进，EOL 已经不再是个问题，有关字节流的滑动窗口机制及其多种优化的算法使得 TCP 已经相当健壮。

11.总结与展望

从互联网架构设计目标的优先顺序来看，互联网架构是成功的。而且 TCP/IP 协议已经在各个领域广泛应用并且衍生出了很多相似的架构。同时它的成功是基于特定优先次序的设计目标来认定的，而且设计者优先考虑的并不是满足实际用户的需要，比如计费统计管理以及分布式运维管理等问题需要设计者更多的关注。

数据报很好地达成了大多数重要的设计目标，但是当我们试图完成后续一些目标的时候，发现计费统计资源管理很难在基于数据报的环境下实现。如前文所述，多数数据报是从源端到目的端传送的一系列数据包的一部分，无法区分应用层次上独立的单元。然而，这一系列数据包对网关是透明的，因为网关独立处理每个数据包。因此，必须对每个数据包独立进行计费统计资源管理，由于丢包重传等对网关透明无法对应用实际传输的数据量进行准确统计。在网络层利用数据报模型使得网络层丧失了重要的信息源，也就很难达成计费统计资源管理等这些目标。

这说明在下一代网络架构中可能有比数据报更好的抽象封装方式。不论采用任何特定的服务类型，它都可以标识从源端发送到目的端的一系列数据包，可以用“流”来描述这种抽象封装方式。这种方式需要记住通过网络的数据流的特征属性，但这却挑战了网络中间设备不记录通信状态信息的架构原则。同时如前文所述，网关的分布式管理控制与互联网协议越来越臃肿复杂的现状，让我们开始考虑集中管理控制方式，以及基于“流”的抽象封装方式，这从根本上挑战了互联网架构。这么理解的话，SDN 和 OpenFlow 似乎在互联网架构变革的路上，数据面和控制面分离的方法也为基于“流”的抽象封装方式提供了技术支撑；部分地颠覆分布式原则的集中管理控制（网络局部）策略又为记录基于“流”的通信状态信息带来了方便，因为基于云计算的数据中心可以高效快速地存储和处理流状态信息，从而使得网络更加智能。

移动性和云计算对互联网架构产生的冲击更具颠覆性。一直以来互联网设计背后都隐藏着一个非常基本的假设：网络和主机都是静态的，换句话说在地理位置上主机是相对固定的，主机所接入的网络也是固定的。互联网架构在两个相对固定的主机（客户端和服务端）及所接入的网络之间提供通信服务。而云计算技术使服务端不再是一个固定的主机，而是根据负

载等各种条件不断变化的一个或多个主机集群，比如某客户端正在使用某个云计算服务，该服务可能在 A 数据中心的某主机（或虚拟节点）上，这时由于负载等条件的变化该服务被迁移到 B 数据中心的某主机上，在这种情况下如何保证该客户的业务连续性？使用 VLAN 的大二层方案似乎可以解决这个问题，但面临可扩展性难题；使用隧道技术似乎能进一步解决可扩展性问题，但这是在现有互联网架构上的修修补补，有没有更理想的解决方案还值得探索。

移动性带来的冲击也与此类似。随着智能手机等移动设备的普及，移动互联网热潮大势所趋。问题是随着地理位置的变化所接入的网络会发生变化，这从另一个方向挑战了网络和主机都是静态的假设。移动 IP 等相关漫游技术在为此努力，可显然它们还没有跳出静态性的假设。

显然，移动性和云计算造成了网络的实时动态变化，而当前的互联网架构并没有为这种动态性作过多考虑。基于“流”的抽象封装方式和 SDN 的思想或许为解决这诸多问题提供了一个可能的思路，但未来以动态性作为基本假设重新审视互联网架构或许是必须的。

参考文献

[1]D. D. Clark, The Design Philosophy of the DARPA Internet Protocols, SIGCOMM 1988

[2]OpenFlow Whitepaper:Software-Defined Networking:The New Norm for Networks,Open Networking Foundation,April 13, 2012

