学士学位论文

基于 IPv6 的 HTTP/2 过渡应用的设计与实现

学　　号: **20151001270**
姓　　名: **陈立翔**
学 科 专 业: **计算机科学与技术**
指 导 教 师: **张峰 副教授**
陈伟涛　副教授
培 养 单 位: **计算机学院**

二〇一九年五月

学士学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师指导下独立进行研究工作所取得的研究成果。除了文中特别加以标注引用的内容外, 本论文不包含任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　年　　月　　日

学位论文使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士学位论文评选机构将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1. 保密 □，在_____年解密后适用本授权书。
2. 不保密 □。

（请在以上相应方框内打 “√”）

作者签名：　　年　　月　　日

导师签名：　　年　　月　　日

摘　　要

今天，网络早已成为人们生活中不可缺少的部分。同时随着我国网民人口的不断增长，互联网环境正在变得愈发复杂。在服务端，开发和运维人员需要利用有限的服务器资源应对更大的并发请求和越来越多的潜在的网络攻击。在这种情况下，原有的一些网络协议已经无法满足要求。目前，IPv4 的公网地址资源已经枯竭，早在 2011 年亚洲地区就已经无法分配到 IPv4 地址；而 HTTP/1.x 协议经过若干次修订后依然被诟病效率过低和功能不足。RFC 发布的 IPv6 和 HTTP/2 协议相关标准已经在很大程度上解决了这些问题，如何将旧有的 Web 应用过渡支持 IPv6 和 HTTP/2 是我们目前面临的重要挑战。

本文对于如何让现有的 IPv4 环境下仅支持 HTTP/1.x 协议的 Web 应用能够在 IPv4 和 IPv6 双栈环境下工作并支持 HTTP/2 协议，同时尽可能提高服务端的并发性能，开展了以下研究工作：

1. 通过对 Go 语言技术的深入研究，分析其协程和并发模型的技术特点。Go 语言是一门在语言层面实现了协程的语言，其实现能够避免操作系统调用产生的开销，单机轻松构建百万级协程，CPU 资源的利用率极高，是近年来服务端开发中解决高并发问题的重要突破点。

2. 比较了 RFC 标准中 HTTP/1.x 和 HTTP/2 报文的区别，分析了如何将 HTTP/1.x 的报文加工处理成 HTTP/2 的报文。同时结合 Go 语言中原生的 net/http 和 tls 库对报文增加 HTTPS 加密，从而兼容目前主流浏览器的实现。

3. 为了将 HTTP/1.x 应用过渡支持 HTTP/2 , 本文采用 Go 语言技术设计了一个 HTTP 网关。该网关通过反向代理技术，在无需对原有应用进行任何修改的情况下，将其升级为支持 HTTP/2 协议，并可在 IPv4 和 IPv6 双栈环境下工作。

4. 基于加权轮询算法和一致性哈希算法对该网关设计了负载均衡策略，使得该网关能够支持横向扩容的 Web 应用，应对更复杂的并发情况。

5. 对该网关进行了压力测试，对比目前市场上提供反向代理功能的服务器如 Apache，Nginx 等，得出该网关在一些情况下能够提高系统并发数和减少响应时间，提升用户体验。并通过火焰图分析该网关在实际工作环境中的性能瓶颈。

综上所述，本文设计的基于 Go 语言的 HTTP 反向代理网关，能够作为在 IPv4 和 IPv6 双栈环境下的 HTTP/2 过渡应用，并且具有较好的并发性能。经过测试，具有一定实用价值。

关键词： IPv6 协议；HTTP/2 协议；HTTP 网关；反向代理；负载均衡；Go 语言；

目 录

第一章 绪论

1.1 研究背景及其意义

各种网络协议常常被称为互联网时代的基石。随着网络环境的变化，网络协议也需要适应时代的需求迭代升级。

从 1996 年开始，一系列关于 IPv6 协议的 RFC 标准发表出来，旨在取代原有的 IPv4 协议。IPv6 的地址长度为 128 位，能提供远多于 IPv4 的地址数量。在 IPv6 环境下，将不需要 IPv4 环境下为了解决地址数量不足而衍生出的 NAT 等技术，每个设备都能得到一个公网 IP，有利于物联网等技术的发展。但 IPv6 技术的推进还需要一段时间，在未来一段时间内我们都将面临着 IPv4 和 IPv6 共存的互联网环境。

2015 年发布的 HTTP/2 协议则旨在取代 HTTP/1.x 协议。为了提高网络传输的效率，更有效地利用网络资源，该协议增加了二进制分帧，多路复用和头部压缩等新的特性。同时目前浏览器中 HTTP/2 实现都强制要求数据通过 HTTPS 加密，而不是像 HTTP/1.x 协议那样将数据明文发送。这显著增强了数据传输的安全性，有效避免数据被劫持等安全隐患。

将现有的应用迁移升级到这些新协议上将会是一个漫长且耗资巨大的过程。如何在今天这种多种协议共存的情况下，将旧有的应用以尽可能低的成本过渡迁移到新协议上，且具有较好的性能？

目前国内外最广泛使用的 HTTP 服务器 Apache 和 Nginx 都支持反向代理并升级 IPv6 和 HTTP/2 协议的功能。通过配置都可以支持通过反向代理将旧有的服务升级至支持 IPv6 和 HTTP/2 协议。

1.2 国内外相关工作介绍

目前国内外市场上 90% 以上的反向代理 HTTP 网关服务器为 Apache 和 Nginx。这两者都基于 C 语言编写。也有很多人对这些网关进行了二次开发，例如中国工程师章亦春等人基于 Nginx 开发的 OpenResty[1] 等。

这些网关在不断的迭代过程中也陆续加入了 IPv6 和 HTTP/2 协议的支持。它们由于久经实际生产环境考验，具有较强的稳定性。但由于一些历史遗留原因，也或多或少存在诸如开发效率较低，并发能力较差等问题。

1.2.1 Apache

Apache HTTP Server（简称 Apache）是 Apache 软件基金会维护开发的一个开源 HTTP 服务器软件。它自 1995 年诞生以来，一直是世界上最被广泛使用的 HTTP Server。同时其也提供反向代理功能，并已加入 HTTP/2 和 IPv6 的支持。

Apache 在二十多年间已迭代数个版本，具有极佳的稳定性和大量的插件，能够帮助开发者快速构建小型 Web 应用。

但是 Apache 在应对并发请求时，使用的是传统上基于进程或线程模型架构，通过多线程来处理并发请求。事实上这种模型虽然稳定，却无法发挥出现代计算机的性能，后面在讨论 Go 语言的并发模型时将会谈到这一点。

1.2.2 Nginx

Nginx 是俄罗斯工程师 Igor Sysoev 在 2004 年发布的一款异步框架的高性能 HTTP 服务器，常被用作反向代理和负载均衡器。目前绝大多数高并发量的网站都使用 Nginx 作为 HTTP 服务器。它在不断的迭代中也陆续加入了 IPv6 和 HTTP/2 的支持。

Nginx 的出现就是为了解决 Apache 的并发模型无法解决的 C10K 问题。C10K 指的是能够使单机服务器承受一万以上的并发连接请求。Nginx 创造性地将模块化加入了设计之中，非常方便于二次开发。如由中国工程师章亦春等人开发的 OpenResty，阿里巴巴公司开发的 Tengine 都是根据实际需求基于 Nginx 二次开发的 HTTP 服务器。

Nginx 采用了 epoll 多路复用机制来应对高并发请求。epoll 是 Linux 2.6 版本内核之后加入的特性。Linux 提供了三种 I/O 多路复用模型，select, poll 和 epoll。它们分别是基于数组，链表和哈希表保存文件描述符的 I/O 多路复用模型。显然 2.6 版本加入的 epoll 模型所使用的哈希表具有最低的时间复杂度，这可以大大减少操作系统切换上下文的开销，应对高并发连接。根据官方的测试结果，在理想情况下，Nginx 单机可以应对五万个并发连接，而在实际的运作中，可以支持二万至四万个连接。

但是这也给 Nginx 带来了一个问题. 在非 Linux 系统下, 无法使用这些操作系统底层的 API 来进行 I/O 多路复用，这时其并发性能会大大下降。对于情况较为复杂的跨平台服务端解决方案来说，Nginx 并不合适。

1.3 论文主要工作和章节结构

本文主要针对如何在 IPv4 和 IPv6 双栈环境下将旧有的 HTTP/1.x 的 Web 服务过渡升级支持 HTTP/2 设计了一个网关。该网关通过反向代理的形式升级协议。并且该网关具有高并发和易跨平台的特性，避免了上述解决方案的一些缺点。

论文的内容和章节结构如下：

1. 第一章：绪论。主要介绍了研究背景，以及通过反向代理升级 IPv6 和 HTTP/2 协议的这方面国内外的一些成熟解决方案及其存在的问题，以及本文主要工作安排。

2. 第二章：Go 语言服务端相关技术介绍。简单介绍了 Go 语言的背景和一些内置库实现，并分析了其并发模型，说明了 Go 语言应用在本项目的一些优势。

3. 第三章：HTTP/1.x 到 HTTP/2 的转换实现。深入分析了 RFC 标准中 HTTP/1.x 协议和 HTTP/2 协议的差异，根据这些差异拟定了程序中通过怎样的流程去升级 HTTP 协议。同时也对广泛使用的 Websocket 协议做了处理。

4. 第四章：IPv4 和 IPv6 双栈环境下反向代理功能的实现。

5. 第五章：性能测试。

第二章 Go 语言相关技术介绍

2.1 Go 语言简介

Go 语言（简称 Go）是由美国谷歌（Google）公司发布的一种静态强类型、编译型、并发型并具有垃圾回收功能的编程语言，于 2009 年推出最初版本并开源。其作者 Rob Pike 和 Ken Thompson 等人均曾经在美国贝尔实验室工作，参与了 C 语言和 Unix 系统的开发工作。

Go 的发明是为了解决 C++ 在服务端开发时的一些问题。在高并发环境下，C++ 由于其过多的语言特性，导致编写并发程序较为复杂。同时其较为落后的包管理机制不利于对代码的高效复用。依赖于系统底层 API 的网络操作等也使得其跨平台能力不强。

Go 本身语法十分接近 C 语言，甚至比 C 语言还要更简单一些，其关键字数量比 C 语言还要更少。对于有一定 C 语言开发经验的开发者来说学习成本较低。Go 内置大量网络编程相关的库，可以大大简化网络编程的工作，更重要的是，其实现不依赖于任何操作系统底层 API，这使得 Go 编写的程序可以较为容易地交叉编译并在多平台运行。

Go 程序以包的形式组织。包实质上是一个包含 Go 文件的文件夹。包内的代码共享相同的命名空间，为区分包内包外的私有成员，Go 将以大写字母开头的对象（包括函数，变量等）对其它包可见，其它的则不可见。

2.2 Go 语言的并发模型

Go 语言最为重要的特性来源于其在语言层面对并发的支持。这里讨论的是 Go 1.7 版本及之前的经典并发模型。虽然在后期版本增加了一些新的功能或语法糖，但本质上还是离不开我们后面讨论的这些并发模型。

2.2.1 协程（Coroutine）

我们知道，线程和进程都是在操作系统层面上实现的。相对于进程来说，线程可以减少调用时的上下文开销，具有更小的颗粒度。

2.2.2 通道（Channel）

2.2.3 上下文（Context）

第三章 HTTP/1.x 到 HTTP/2 的转换实现

要想实现 HTTP/1.1 到 HTTP/2 的转换，我们首先要了解这两个协议的特性及其异同，针对性地设计转换的方法。

世界上最早的 HTTP 协议标准源于 1991 年发布的 HTTP/0.9 标准。该版本及其简单，只有一个 HTTP 方法 GET。它具有以下特点：

- 客户端/服务端响应都是 ASCII 字符
- 客户端请求由一个回车符（CRLF）结尾
- 服务器响应的是超文本标记语言（HTML）
- TCP 连接在文档传输完毕后断开

显然这个协议是十分简陋的。它无法处理用户登录等稍微复杂一点的操作，以及各种富文本。1996 年 RFC 发布了正式的 HTTP/1.0 标准，相较于之前的版本它有一些明显的特性：

- 服务端响应增加了响应状态
- 请求和响应增加了多行首部
- 响应内容不再局限于 HTML

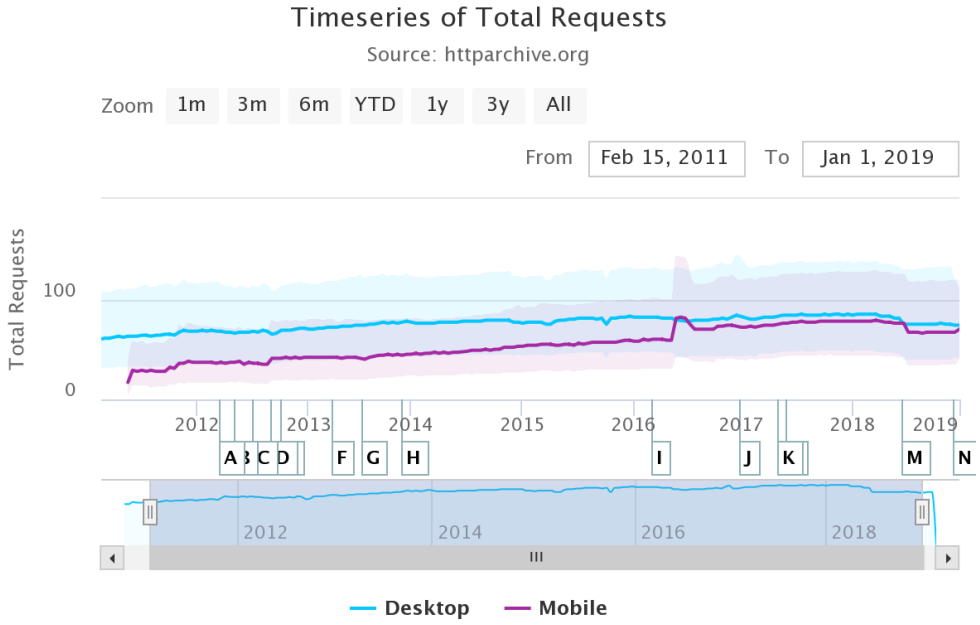
HTTP/1.0 依然存在很多问题和值得优化的地方。比如对 TCP 连接的复用，大文件传输的优化，浏览器缓存较差等等。1999 年发布的 HTTP/1.1 就是为了解决这些问题的。该协议一直到 2014 年，依然在不断地进行修订。它具有以下几个显著特性：

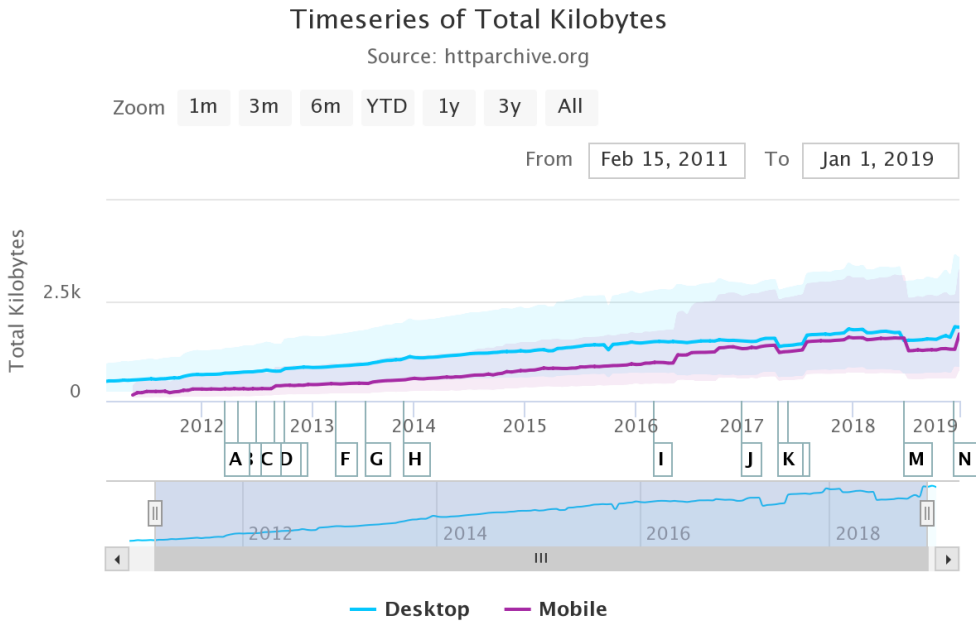
- 持久连接（Keep-Alive 机制）
- 传输编码（Chunk 机制）
- 字节范围请求（Accept-Ranges）
- 增强的缓存机制（协商缓存和强缓存）

HTTP/1.1 已经有二十年的历史，并且依然在被广泛使用，十分稳定。但 2015 年，RFC 依然发布了 HTTP/2 用于取代 HTTP/1.1。这源于互联网快速的发展，使得其无法适应当前的需求。

3.1 HTTP/1.1 的困境

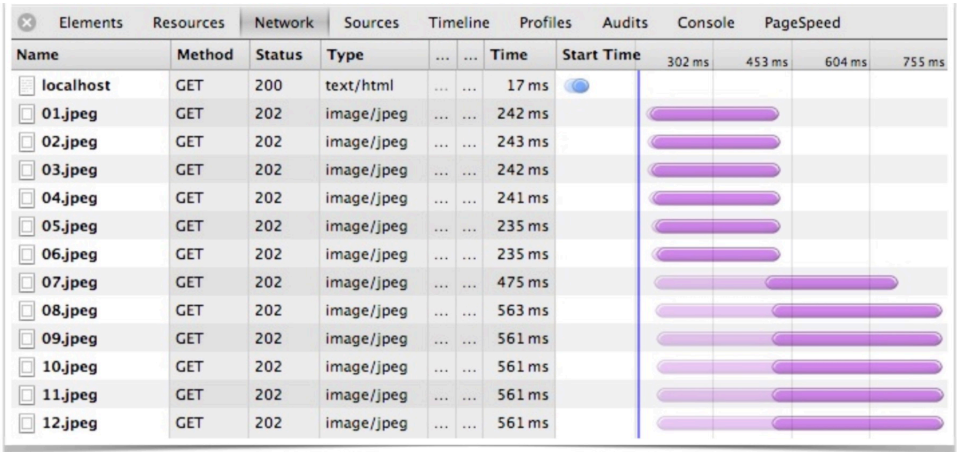
如今的 Web 应用已经大大超出了人们在最开始对 Web 的想像。在早期，一个网站往往只有一个简单的 HTML 页面。但是在今天，任意一个门户网站都有大量的流媒体，AJAX 请求，图片……人们也不再仅仅使用 PC 来浏览网页，移动端和物联网设备逐渐成为了主流。这种情况下，一个 Web 页面中的 HTTP 请求数和网站所占用的通信和存储空间会大大增加。httparchive.org 对目前互联网的的大量网站做了监测，结果如图所示[2]。





从图中可以看出，随着时间的推移，浏览一个网站平均需要的 HTTP 请求数量和资源的大小都在稳步上升。目前在没有缓存的情况下，在 PC 端浏览每个网站平均需要下载 2000kb 以上的资源，发生 70 个以上的 HTTP 请求。在移动端这个数字还要更高。这给了网络传输以很大的挑战。事实上，用户浏览网站时，在等待的过程中，平均有 69.5% 的时间是耗费在网络操作的阻塞上的。

而 HTTP/1.1 协议本身对并发连接并没有做很好的优化。Keep-Alive 机制本身只能复用连接，但是对并发连接并没有什么帮助。早期的标准 RFC2616 甚至规定浏览器中同域名只能有两个连接。RFC7230 虽然去掉了这一限制，但现代浏览器中依然限制同域最多只能有 6 个并发连接。在浏览器中打开一个 HTTP/1.1 的网站，我们很容易在控制台中看到网络资源的加载时序，如图所示。



3.2 HTTP/2 的新增特性

针对上文提到的 HTTP/1.1 的问题，2015 年 RFC 发布的 HTTP/2 协议新增了若干特性：

- 引入了流（Stream）的概念，取消了 Keep-Alive 而采用多路复用机制。
- 引入二进制分帧，将消息分割为更小的帧，并采用二进制格式进行编码。其中头（Header）会被专门封装到 Headers 帧中。
- 引入头部压缩机制，将一些常用头部用单字节表示，缩减头部大小。

这些机制解决了并发连接的问题，并在一定程度上缩减了每次 HTTP 传输的大小。

特别需要注意的是，HTTP/2 目前有 H2C 和 H2 两种实现类型。H2C 是目前主流浏览器所实现的类型，该实现较为强调传输的安全性，强制通过 TLS 加密传输的数据。H2 多用于非 Web 的应用场景，例如 RPC 调用等。下文所讨论的均默认为 H2C 的实现。

3.3 HTTP/2 协商机制

在 HTTP/1.1 和 HTTP/2 共存的环境下，浏览器和客户端要怎么样才能知道到底应该选择哪个协议来沟通？毕竟还有很多低版本浏览器不支持 HTTP/2。

3.4 WebSocket 支持

3.5 HTTPS 支持

第四章 IPv4 和 IPv6 双栈环境下反向代理功能的实现

4.1 用户配置

4.2 反向代理的实现

4.3 负载均衡策略

4.3.1 加权轮询算法（Weighted Round Robin）

4.3.2 一致性哈希算法（Consistenting Hashing）

4.4 IPv6 环境下工作

第五章 性能测试

5.1 压力测试

5.2 火焰图

致谢

参考文献

[1] SCHWARTZ M, MACHULAK M. Proxy[G] . Securing the Perimeter. [S.l.] : Springer, 2018 : 205 – 229.

[2] HTTPARCHIVE. The number of resources requested by the page.[R]. .