流撕裂者：一种自适应的分布式局域网缓存

陈文迪、陈文浩、游灏溢

2021/12/12

1 项目简介

随着互联网的发展与技术的更新迭代，网络已经进入人们生活的方方面面。每天每位用户会产生海量的互联网请求，如浏览网页、下载文件或是观看直播和视频，这些互联网服务极大方便了人们的生活。然而，互联网体验中一个主要的问题是卡顿与延迟。卡顿产生的原因有很多种，但大多情况都是因为网络带宽有限: 由于网络中的请求过多，每个网络请求不能被分配到足够的带宽，从而引起卡顿。卡顿不但会增长资源下载时间，而且会降低人们网络体验，例如降低视频画质。就此问题，我们团队提出了一种自适应的分布式局域网缓存。

在实际调研中我们发现，网络负载过大的发生常常与短时间内高并发的下载文件、观看视频请求有关，而这些情况往往发生在同时间大量同学有相同需求的时候，例如课堂课后获取课程资料，以及在线赛事、节假日期间同学们同时观看视频和直播。就在英雄联盟 S11 总决赛当晚，全校同学大约有 1/3 都在观看比赛，这对校园网产生了极大的负荷。

在这些情况下，同学们的需求往往是相同的（如比赛直播资源，课程文件资源）。对于相同的资源，每个人重复请求是对网络资源极大的浪费。此外，虽然对互联网的访问速度是受限的，但是局域网内的传输速度一般较快。目前，基础网络设施已经提供了本地网络缓存，但是小范围、细粒度的局域网缓存服务并未普及。如果我们能利用局域网内高速的传输速率，整合同学们的需求，共同向服务器请求一次，再分发给每个需要资源的机器，就能极大加速网络传输效率，提高互联网带宽的利用率。基于这个想法，我们提出并实现了一种自适应的分布式局域网缓存，并使用 bilibili.com 提供的视频流服务作为测试对象测试了其缓存性能。此外，针对流媒体对象分块不一致的问题，我们提出了一种“整流—重组”的算法。我们将整个框架称为流撕裂者（StreamRipper）。

2 设计架构

2.1 原理简述

本项目主要针对互联网流媒体访问这一场景展开优化，利用本地局域网缓存加速流媒体访问。其基本原理如下：假设集群内的用户所访问的数据是相似的，并且集群内的局域网带宽远大于互联网访问带宽。依据假设，在局域网内建立共享集群，对于集群内用户请求过的数据，我们将其缓存在局域网缓存中，此后其他用户若访问同样的数据，将直接从局域网缓存中获取。利用局域网内部速率与对外速率的巨大差距，局域网缓存可以加速用户的流媒体访问速率，减少对外互联网访问流量，最大化带宽利用率。

局域网缓存就其本质来说提供的是一种存储服务，并且为了实现负载均衡和较好的可扩展性，这种存储应该是分布式的。此外，为了使得缓存对用户是透明的，我们所设计的局域网缓存应与现有的网络协议框架所兼容，具体来说，我们需要分析截取用户正常的互联网请求，并依此查询缓存，当缓存命中时将相应的数据返回给用户。最后，我们还需要考虑性能问题，例如如何选择下载节点，如何解决流媒体片段不一致的问题。本章的后几个小节将针对分布式存储、请求拦截修改与性能调优这三个挑战给出我们的解决方案。

2.2 项目整体框架

为了解决2.1中所描述的三类挑战，我们按功能划分将整个框架划分为了三块：前端，中间件和后端。如图 1 所示。通过采用分块开发的策略，每个模块仅与相邻模块发生数据交换，可以极大地减少调试难度，并且提高了整个系统的扩展性。

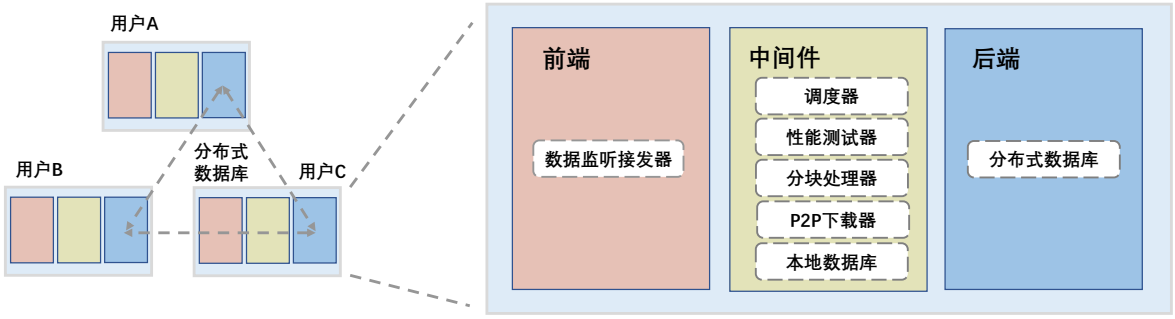

图 1: 项目整体框架

前端主要是一个数据监听接收发送器。它通过对本机的网络请求的监听，拦截程序的请求，并对请求进行分析，将数据请求交给中间件处理。中间件在收到本机的请求后，将进行相关的调度、查询工作，将请求数据返回给前端。最后，前端将接受到的数据进行处理并转发给请求程序，如浏览器。虽然数据的请求和接受通过了前端的处理，但在浏览器看来，它并不知道该程序的存在，对于用户而言，就像是和从服务器端访问一样自然。

中间件分为五块，分别是调度器，性能测试器，分块处理器，P2P 下载器和一个本地数据库。调度器根据前端发出的网络请求，查看后端的分布式数据库，对于未命中的请求，将在集群中选择一个合适的节点用于下载。而性能检测器会定期监测更新本机网速，将信息写入后端的分布式数据库，从而用于调度器的决策。由于不同用户请求同一流媒体资源时，所请求的数据包分块不是对齐的，而分块处理器则通过再分块-重组的方式解决这一问题。P2P 下载器和本地数据库用于在集群内部传输和保存数据，当缓存命中时，调度器将通过后端的分布式数据库查到存有该数据机器的 IP 地址，并通知 P2P 下载器从该机器的本地数据库中下载所请求的内容。

后端主要由一个分布式数据库构成。该数据库记录的并非具体的视频流数据，而是每个数据所存放的位置，以及性能测试的结果，是整个集群的基础。由于分布式数据库可能被集群中的任一节点所修改、读取，我们采用 Raft 协议来确保数据库数据的一致性。在本项目中，我们使用开源、轻量级的 rqlite 作为分布式数据库解决方案。

2.3 前端设计

对请求的透明拦截是用户体验的核心，我们的设计目标是当用户开启分布式局域网缓存后，其互联网访问体验与直接向服务器请求时一样自然、流畅。这是本项目区别于局域网投屏共享的主要区别。

用户正常向服务器发送请求时，需要经历完整的五层协议栈，而利用本地代理，我们可以跳过部分协议栈，充分利用缓存性能，因此我们提出了方案一：利用 HTTP 代理，直接拦截 HTTP 请求。其拦截流程如下：当用户发送 HTTP 请求报文时，由我们所设计的 HTTP 代理拦截该请求，若

缓存命中，则直接由代理服务器向浏览器发送回复报文。然而，在代码编写的过程中，该方案遇到了明显阻力：现代流媒体服务采用的是 HTTPS 的安全协议。如图 2所示，HTTPS 综合利用了非对称加密和对称加密，这使得数据的传输并不是明文的，这也就表明代理服务器作为第三方无法随意读取、篡改数据。在前期调研过程中，我们发现现有的代理服务确实对 HTTPS 提供支持，其基本

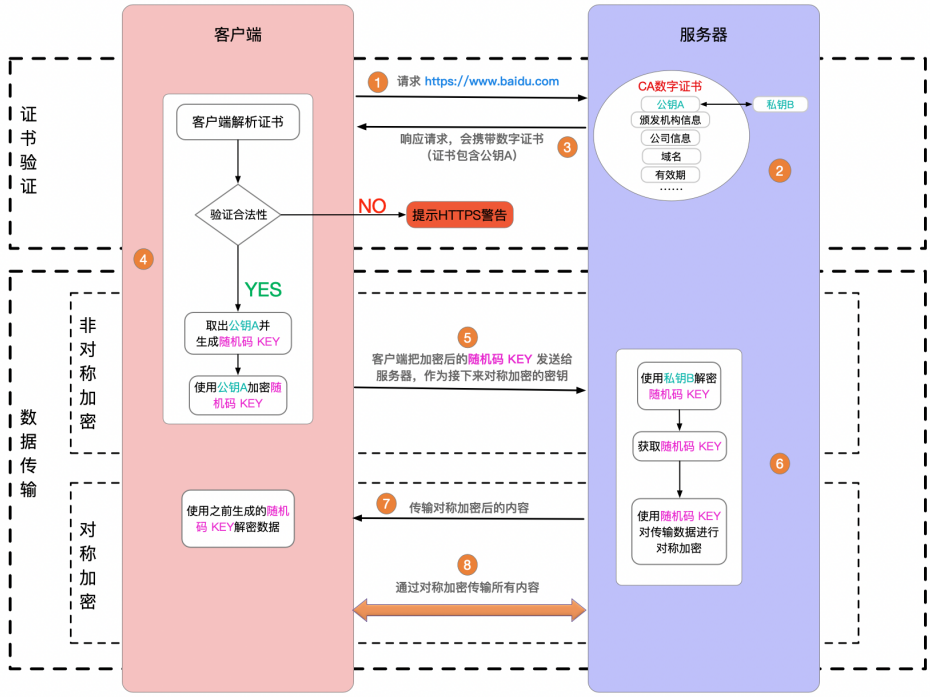

图 2: HTTPS 加密流程

原理都是在传输层对 TCP 包进行一定的封装。由此，我们提出了方案二：将代理层级从用户层下降到传输层，直接对 TCP 包进行拦截和修改。然而，这个方案也是不可行的：这是因为传输层本身所包含的信息太少，HTTP 报文中的请求 URL、视频分块等关键信息仍然是加密的，我们便无法利用这些信息设计缓存。并且，加密的 HTTP 报文也无法被不同设备所解密，此方案在可行性上仍有缺陷。

然而，HTTPS 并不是完全无法修改。由于目前的 HTTPS 仍然是基于 CA 证书的，若我们可以伪造 CA 证书，便可以通过中间人攻击的方法截取 HTTP 报文并撰写回复报文将指定内容交付给浏览器。在进一步调研中，我们发现，这种基于中间人攻击的代理是存在的，在本项目中，我们选择 mitmproxy 作为代理服务，并依此提出了方案三：如图 3所示，通过让浏览器信任我们所伪造的 CA 证书，这样便可以让客户端详细其正在与服务端通信，这样代理服务器可以分别于客户端、服务端建立 TLS 连接，而此后所有客户端的请求与回复报文对于代理服务器便是明文的。

综上，前端的数据监听解发器主要由一个基于中间人攻击的代理服务器实现。如图 4通过这样的设计，我们对用户请求的拦截可以全部在运输层之上完成，当缓存命中时，便可跳过其他的网络协议栈而直接获得回复报文，最大化地利用了缓存性能。

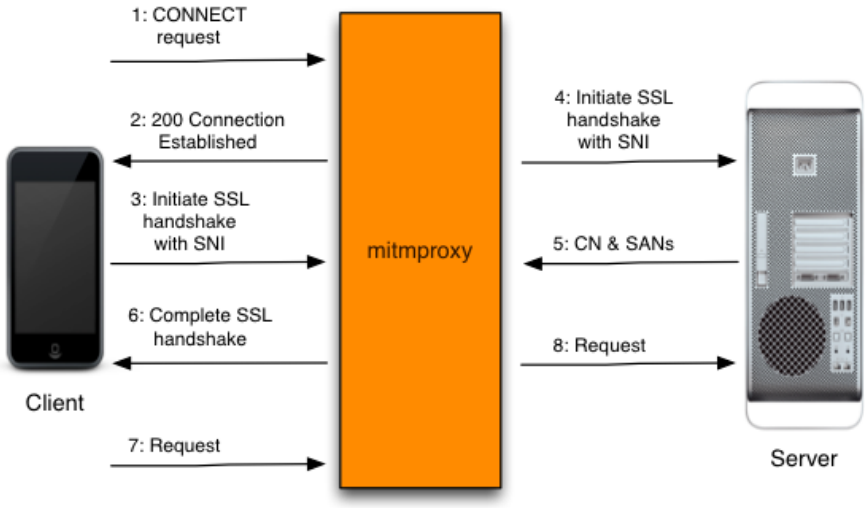

图 3: mitmproxy 实现 HTTPS 中间人攻击原理图

2.4 中间件设计

本项目中，丰富的中间件为前后端的协同提供了强大的支持。下面我们将中间件的五个主要组件分成三个模块：调度器与性能测试器，分块处理器，P2P 下载器与本地数据库来阐述我们的设计细节。

调度器与性能测试器 调度器的设计目标是，当缓存不命中时，需要挑选一个合适的节点委托下载任务。为了兼顾性能与负载均衡，我们需要考虑两个方面的因素：主机的互联网访问速率与当前已被委托的数量。因此，我们设计了如下的打分函数，来为集群中的每个节点打分：

$$score(i) = speed(i) + \frac{\alpha}{delegation(i)}$$

其中 $speed(i)$ 是节点 i 的互联网下载速率，通过性能测试器获取，$delegation(i)$ 是节点 i 已经被委托的任务数量，α 是一个负载均衡调节因子。当由任务请求时，调度器将查询数据库中已记录的每个节点的性能数据，再依据打分函数选择分数最高的节点委托新的下载任务。此外，如果集群中每个节点都有可能称为被委托方，则我们的架构中有可能发生死锁，即 A 的委托被调度给 B，而 B 的委托被调度给 A。因此，我们需要给集群中的每个节点一个类别：server 或 client，server 节点所有请求将不经过调度器，而只有 server 节点会进行性能测试并成为被委托方。

最后，我们简单阐述一下性能测试器的设计。在本项目中，我们采用开源的 Speedtest CLI 工具用于测试每个节点的互联网下载速率。性能测试器将作为一个独立的线程，定期选择最优服务器进行速率测试，并将结果写入分布式数据库中供调度器使用。

分块处理器 视频流的分块优化了视频传输的过程，但也给本项目中已下载视频块的处理带来了困难。由于服务器端分块的不确定性，就算两个用户观看相同视频的，从相同时间开始，也有可能因为网络状态的差异，下载具有不同终止范围的视频块。如果我们直接将这些原生的视频块存入数据库，很有可能造成本地数据库中存储了大量无法完全匹配但又有相重叠部分的视频块，这既降低了空间利用率也极大地奖励了缓存的命中率。因此，我们引入了分块处理器对视频的下载请求进行了重新分块，如图 5所示。

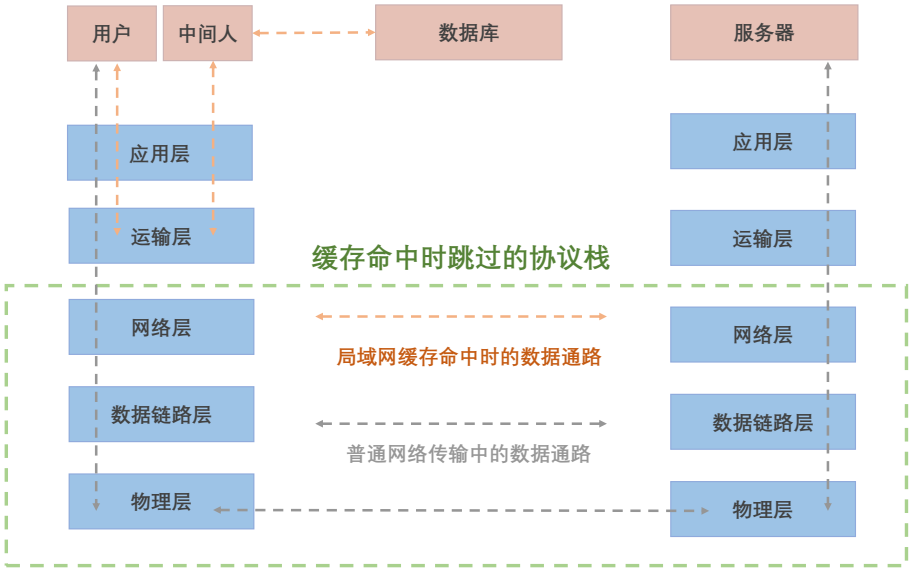

图 4: 开启代理服务器时数据通路的变化

HTTP1.1 请求 headers 中的 range 字段定义了视频块的起止位置，这使得修改请求报文的范围和抽取回复报文的数据成为可能。综合考虑了实现复杂性和引入分块的开销，我们选择 512KB 作为分块的大小，并强制修改每个请求报文的范围为该数值的倍数从而避免插入操作的复杂性。具体来说，对于一个下载请求，我们首先计算出能够覆盖原始范围且符合分块要求的新范围，采用相应的方法处理新范围的视频块，最后将良性分块的视频块插入数据库，并根据原始范围抽取相应的数据返回。分块处理器的引入不仅使得数据库空间利用率和命中率极大地提升，也优化了部分命中情况下的 P2P 下载器的工作负担。

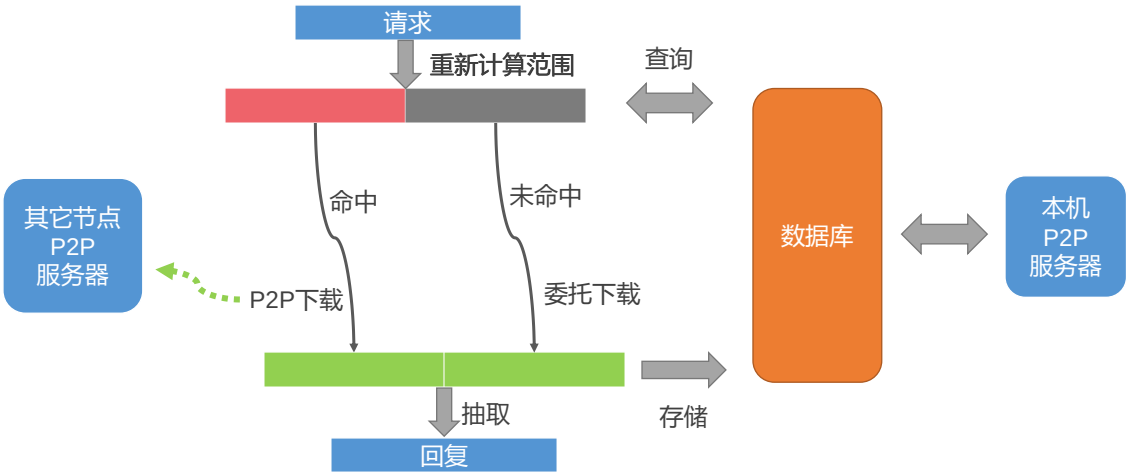

图 5: 分块处理器和 P2P 工作示意图

P2P 下载器与本地数据库 P2P 下载器与本地数据库共同支撑了缓存命中时，集群节点之间的数据传输。我们使用 Python 的异步 io 库 asyncio 实现了 P2P 服务器的客户端与服务端，通过对 socket 的进一步封装简化了实现。每个节点加入集群时，会开启其 P2P 下载器的服务端，并持续监听相应的端口请求。一旦其他节点在缓存命中时通过分布式服务器获得了对应的主机 IP 和查询键值，就可以通过 P2P 下载器的客户端访问对问的主机并获取相应的数据。

本地数据库的实现并不复杂。通过对 OrderedDict 的封装，我们使其支持了线程安全，并且支持了高并发的读取。此外，本地数据的大小并不是无限的，在本项目中，我们使用 LRU(Least Recent Used) 作为表项替换算法。

2.5 后端设计

本项目中，后端的分布式数据库需要保持不同节点之间数据一致并且有一定的错误容忍能力，从而为中间件的提供有效支持。因此，我们选择了实现 Raft 一致性算法的开源分布式数据库 rqlite，并将其封装在后端以方便调用。我们将从集群组建和数据库查询两个方面，进一步阐述我们后端分布式数据库的选择。

在集群组建方面，相比于传统的利用局域网广播实现自组建，rqlite 支持使用 discover id 向官方提供的网页 API 通信以实现节点的自动加入，该方法可以实现在同一个局域网里建立不同的集群，也支持在不同的局域网之间建立集群（本项目中并无体现）。在数据库操作方面，rqlite 提供了原生的 Python API，并且实现了数据库的高并发和原子性。

此外，由于基于 Raft 实现的一致性算法在每个节点上都会保持相同的数据。因此，综合考虑了空间复杂度和错误容忍能力，我们选择在分布式数据库仅存储每个视频块的所在的节点 IP 和性能测试的结果，而不是直接存储视频块，从而避免了

3 项目运行流程

3.1 整体流程

每个用户运行本项目时，由三个主要流程。如图 6所示: 首先，用户作为节点申请加入集群；在申请成功后，开始不断执行代理服务器，代理服务器会完成与局域网代理相关的操作；最后代理服务器捕获到用户的中断请求，结束代理并退出集群。

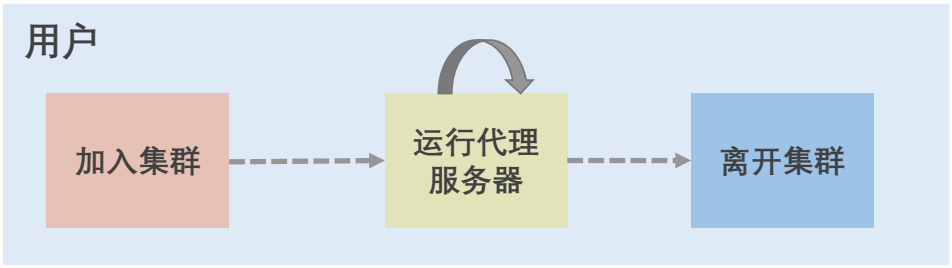

图 6: 项目整体运行流程

3.2 加入与离开集群

如图 7所示，用户加入和离开集群的过程是对偶的。当用户加入集群时，首先它需要确定其所属节点的类型（server 或 client）和加入集群的 discover ID；接着，它需要完成一系列的准备和初始化工作，例如本地数据库的创建、P2P 下载器的启动等。而退出集群时的操作是相反的，当捕获到中断请求时，应首先关闭中间件再退出集群。这样的退出流程保证了单节点的退出不会破坏整个分布式数据库，保障整个系统的稳定性。

图 7: 加入与离开集群的具体流程

3.3 程序运行过程

程序运行主要分为本机性能检测，P2P 服务器和代理服务器三个并行部分，如图 8所示，以下分别说明。

本机性能测试器会隔一段时间进行一次测速，这些信息会被同步到分布式数据库中，以便于调度器查询来分配下载请求。P2P 服务器会监听来自其它节点的请求，并返回本机数据库中相应的视频块。

代理服务器作为最核心的部分，主要负责实时监听用户的网络请求。当收到用户向服务器发送的网络请求后，会进入代理服务器工作流程。代理服务器工作流程如下，对于拦截到的请求，首先将所请求的数据分块并查询。如果全部命中，则分别向存储有相应的数据块的用户请求这份数据，最后将数据整合并抽取后返回给本机。如果不幸未能全部命中，则说明集群中不存在包含该数据的全部数据块，此时应该向服务器申请缺失的数据块。由于所选择的代理架构的限制，代理服务器无法在单次请求的流程中，在本机发出多次不同数据块的请求。因此，由于 server 用户无法委托其它节点下载，server 用户必须自己重新下载全部的数据，然后对集群中不存在的数据块进行补充。对于 client 用户不存在这样的限制，因此 client 仅委托未命中的数据块给 server 用户下载。

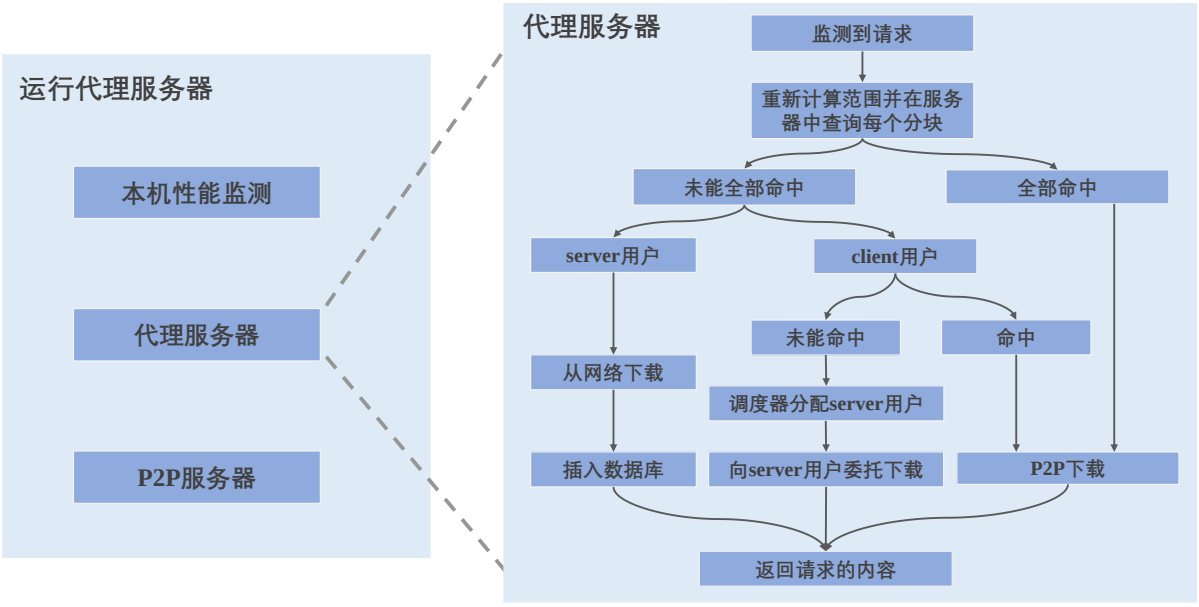

图 8: 代理服务器的具体流程

4 实验与结果

4.1 正确性测试

本小节从四个方面验证实现的正确性：浏览器流媒体视频能否正常播放，回复报文是否被代理服务器修改、调度器能否正确进行下载请求委托以及 P2P 下载器能否正常工作。

首先，进行流媒体播放测试，将浏览器的 HTTP 代理指向代理服务器端口，开启代理服务器，并打开任意视频播放页面。从图 9中可以看出，视频可以正常播放，这说明我们所设计的缓存服务对用户是透明的，用户不需要离开浏览器即可使用局域网缓存功能。

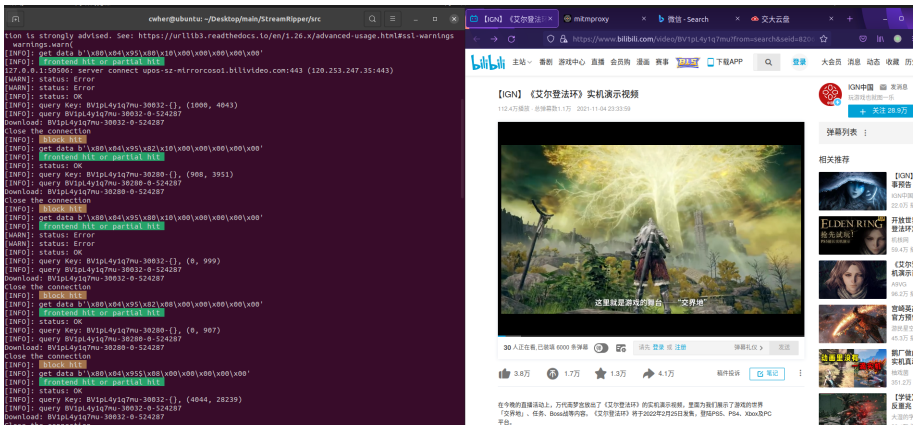

图 9: 使用局域网缓存时的用户体验

然后，我们验证回复报文确实经过了局域网缓存的处理。为此，我们在代理服务器中将回复报文中的 server 字段手动改为 StreamRipper，并查看浏览器所接收到的回复报文。从图 10中可以看到，浏览器所接受到的回复报文确实经过了局域网缓存的处理。

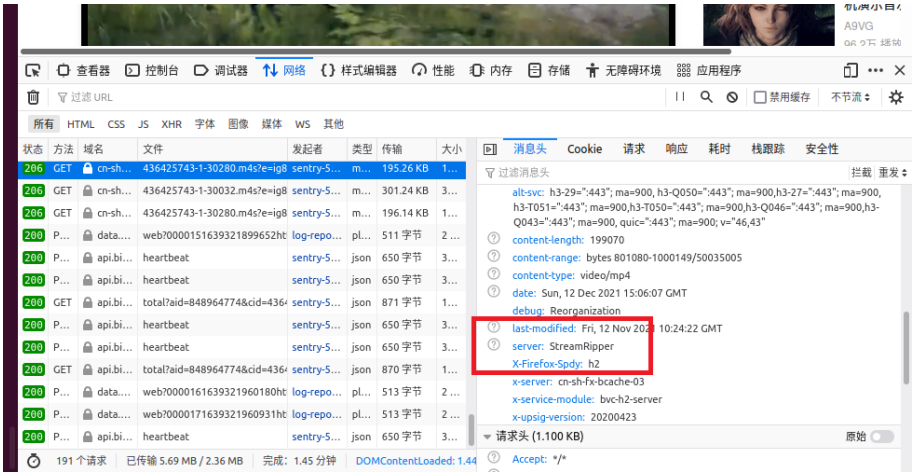

图 10: 使用局域网缓存时回复报文中的 server 字段

接着，我们验证调度器能正确地进行下载请求委托。我们分别查看委托请求放和委托下载方的 log 记录，从图 11中可以看到，委托方请求发送之后，被委托方成功接受了请求，并将委托方所请求数据发回。

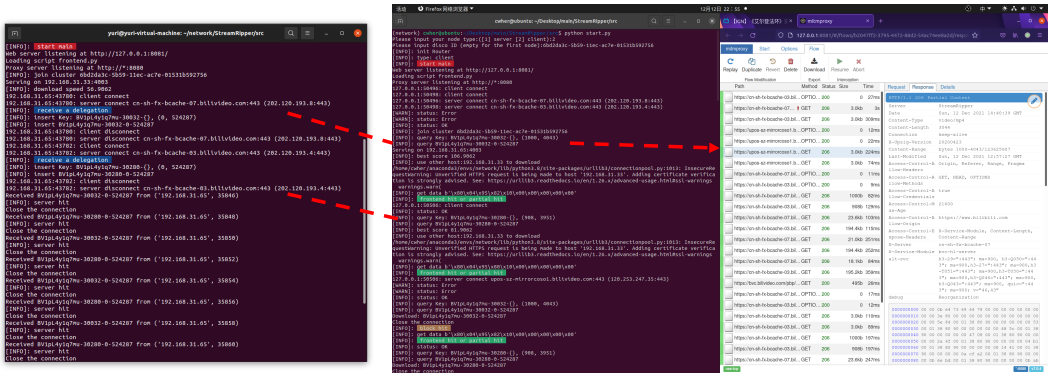

图 11: 委托的请求与处理

最后，我们验证 P2P 下载器的正确性。我们分别查看 P2P 下载器服务端和客户端的 log 记录，从图 12中可以看到，客户端发送 P2P 下载请求被服务端正确处理，并且服务端成功将所请求数据发回了客户端。

4.2 本机缓存性能测试

在这一小节我们对本地缓存的性能进行了测试，并与直接从服务器下载的时延进行了对比。如图 13左侧所示，设置块大小为 512KB 时，直接向服务器请求一个数据块需要耗费 300ms 以上的时间。即使去除分块以及重新计算范围的操作，使用原始的请求范围，一般也需要耗费 150ms 以上的时间。本地缓存的命中则可以极大地降低这一时延，如图 13右侧所示，当本机缓存命中时，一般只需花费不超过 50ms 即可获得数据。此外，对于 P2P 通信获得的数据块，通常也仅需要花费不超过 200ms 的时间。

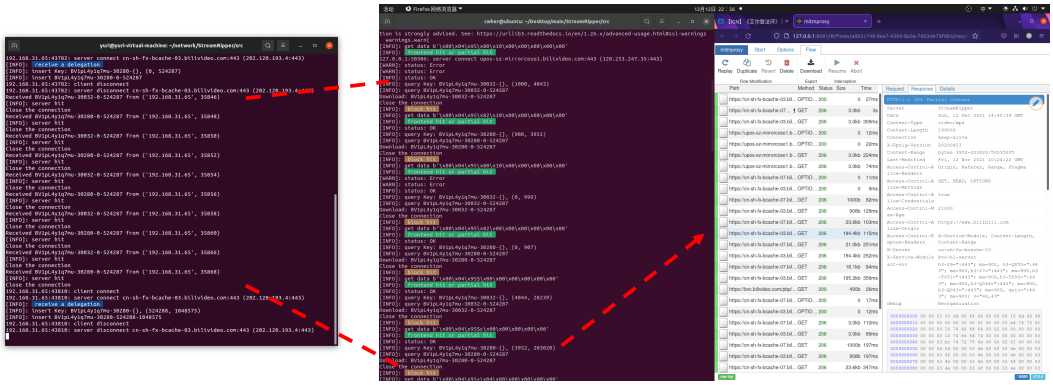

图 12: P2P 请求的发送与处理

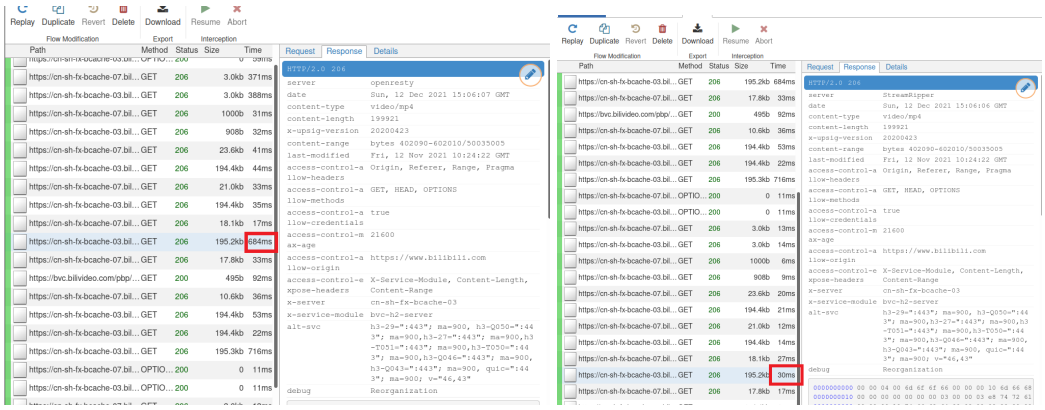

图 13: 本地缓存性能对比

5 总结与展望

本文从生活中的应用场景出发，提出了一种自适应的分布式局域网缓存，并针对分布式存储、请求拦截和性能调优这三个挑战分别给出了我们的解决方案。此外，根据互联网流媒体服务的特点，我们设计了一种分块处理器，有效地减少了数据库冗余并降低了系统整体负载。最后，我们将此项目在 bilibili.com 这一实际环境中进行了多种实验。实验结果显示，本项目可以利用 P2P 下载器和本地缓存减少对外流量请求，可以很好地解决低质量网络环境下的重复请求问题。

当然，本文中所提出的架构仍有优化空间。我们在此给出三种可能的优化方向：通过更智能的分块策略，例如定期交换分块的存储位置，可以实现更好的局部性，带来更好的并发性能；通过对 P2P 下载请求的合并，我们更好地解决局域网内的性能冗余，提升 P2P 传输性能；针对 HTTPS 的拦截问题，可以设计浏览器插件在加密前获取 HTTP 原始报文，从而避免中间人攻击所来带的风险与性能损失。

6 致谢

感谢孔令和教授在课堂上带来的深入浅出的讲解，在大作业开题前与项目组成员交流讨论，促成了本项目的顺利开展。感谢助教学长的辛勤工作。感谢叶航宇、张鼎言等同学在课后与我们讨论。

参考文献

1. **Real-Time Messaging Protocol**:https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol
2. **Proxy Server**:https://en.wikipedia.org/wiki/Proxy_server
3. https://mitmproxy.org/
4. https://github.com/rqlite/rqlite
5. https://raft.github.io/
6. https://segmentfault.com/a/1190000021494676