MulticastReceiver Explainer

Web 直播下的广播组播技术

MEIG+WNIG：联合研究主题

源文链接:
https://github.com/GrumpyOldTroll/wicg-multicast-receiver-api/blob/master/explainer.md

§1 问题与动机

CDN 或其他大规模分发热点直播内容的能力无法满足需求。

扩展问题示例

一些不同的演示文稿解决了可通过 multiast 解决的缩放问题，但在此我们将引用 NANOG 79 上提供的一些实际数字：

- NANOG79 视频

  https://www.youtube.com/watch?v=2aihLUb1elg&t=4m56s

- NANOG79 材料 PPT

  https://storage.googleapis.com/site-media-prod/meetings/NANOG79/2209/20200530_Holland_Ip_Multicast_Next_v1.pdf#page=5

注意：演示文稿使用错误的数字进行视频比特率估算，以下文中进行了更正，但总的来说 NANOG79 推论仍然成立。

- 167 tbps：由 Akamai 于 2020 年 4 月宣布的最终用户交付流量的新纪录。

场景 A：线性媒体（广播电视节目）分发

使用以下生产数据：

- **20 mbps**：4k 视频的比特率
- **5 mbps**：1080p 视频的比特率

可以达到的受众规模示例：

- **835 万用户**= 167 tbps /（20mbps /最终用户）
- **3350 万用户**= 167 tbps /（5mbps /最终用户）

以热门赛事活动的受众规模做对比示例：

整体收视人数：

- 5 亿 = FIFA 世界杯决赛，2018 年
- 2-3 亿 = 2015 年板球世界杯
- 1 亿 =超级碗 2019

在线观看人数：

- 600 万观众= Twitch 并发观众的所有时间均在 2020 年 6 月达到峰值（截至 2020 年 12 月）
- 900 万观众= 2018 年世界杯在线并发观众高峰

什么*是* 4k 的实现为最大上限？

2017-2018 Nielsen 节目”Scorpion”：最受欢迎排名**第 43 位:** 观众数约 830 万。

而 Internet 的单播交付模式下，Internet 上最大直播商之一的能力只能处理第 43 名最受欢迎的电视节目。Internet 确实无法发挥其潜力。

场景 B：内容下载

厂商需要分发给 10 亿终端设备 2GB 大小的操作系统升级包文件。高端峰值交付率下的平均交付时间是多少？

- **1.1 天**= 2GB * 10 亿设备/ 167 Tbps

若某些操作系统或应用程序更新超过 **10GB**，因此以创纪录的总交付速率向 10 亿用户交付将花费超过 **11 天的时间**，以上尚未考虑其它紧急的安全更新任务！

结论

我们认为这些规模问题可以通过 IP 广播/组播技术解决。

如果 Web 能够订阅这种流行内容，并将流量作为通过网络传递的一部分复制到许多最终用户，那么它将对 Internet 处理这些扩展问题的能力产生革命性的影响，并将显著提高提高网络效率，还能服务与比重要热点节目优先级略低的其它直播内容。

（即使现实世界中有时需要这样做，极端情况下今天也无法通过互联网以任何方式令人满意地完成。）

§2 目标

- 提供一种订阅一对多组播流的方法。
- 提供可在多用例场景下使用的 API。多种协议可能包含开放或私有协议。但只要他们能够满足安全规则需求，这些技术都会受到开发者欢迎。
- 加密验证流量。应用程序只能从带有数据包的数据中接收数据，该数据包具有由服务器正确控制的原始策略的加密证明，该数据是由正确的发送者发送的。
- 确保网络安全。使用 API 不能通过注册太多流量来制造网络拥塞，否则将作为风险引起平台监控，并且在浏览器或网络中进行过滤。

§3 非主要目标

- **发送**：此 API 不产生任何出站数据通信。该 API 中的功能调用了一些 IGMP 或 MLD 的出站数据包，以及 IETF RFC 系列中定义的其他信令协议。除了用于加入广播/组播组和处理其流量的特定窄信令之外，不提供创建应用程序可控的出站流量。

- **ASM（Any-Source Multicast 任意源的组播）**：提议仅允许使用 SSM（Source Specific Multicast 特定源的组播）。（注意：RFC 8815 不推荐使用 ASM 进行域间组播）

- 本地网络中的**隐私和保密性**：

  作为接收者，您的 IGMP 或 MLD 成员资格数据包会发送到 LAN，在该本地网络上可查看您计算机的 IP/MAC 地址。

  网络复制数据包，且他们知道源和组的全局 IP。如果他们愿意，他们可能可以找出复制包中所含的内容。

  该 API 主要用于预期有大量用户使用的热点直播节目内容，但是如果您无法让本地网络知道您正在使用它，则必须将其作为加密单播从其他地方进行隧道传输。不只是您和发送者，而且也不是，因为网络参与了数据包的复制。您需要与复制该数据包的下游用户建立信任关系，因为他们可以了解您对节目内容使用情况。

  在本地网络的上游，节目内容数据流变得更加私有。他们只知道网络中至少有一台计算机已订阅，而不知道是谁。但是您可以假设：下一跳路由器将知道您的计算机已要求接收特定内容。

- **数据加密**：

  尽管发送方可以加密数据，但它不是必需的；在可伸缩的一对多分发的预期用例中，它也不能有效地提供隐私或保密性。数据包的内容被发送给许多接收者，其中一些是不可信的。任何允许接收者解密的密钥也可以被接收者的对手使用，可能是因为他们是发送者的合法客户并为此付费。

  这就是说，数据可以例如由加密的段构建而成，并且接收器可以构建那些段并将它们馈送到播放器中，该播放器提供关于密钥控制的自身保证，就像许多现有 DRM 系统所做的那样。

  没有什么能阻止数据被加密，这是完全可能的，而且在某些用例中是可能的。但这种体系结构中没有为传输的数据提供或要求加密。该规范向 web 应用程序提供经

过身份验证的 UDP 有效负载，不同的层必须理解这些有效负载的内容（包括它们是否包含加密数据）。

§4 建议解决方案

概述

该 API 使用 UDP 端口号和认证元数据的域名来订阅特定于源的广播/组播（S，G）。若连接成功（当然可以退出），它将接收 UDP 有效负载。

如果丢包率足够高，或者总带宽超过系统管理的阈值，则连接将失败并显示错误。

如果丢包率高于安全阈值但低于紧急断开阈值，则应用程序将有一个信号和宽限期，可以正常过渡到保持较低丢失级别的状态。。

规范

这个 API 将建立在 IETF 中正在进行的一系列工作的基础上。其基本思想是为浏览器提供一种标准化的、安全的方法来发现和处理有关流量流的元数据，以确保安全运行。（网络也可以使用相同的元数据来确保网络安全。）

- DORMS（针对 SSM，RESTCONF 元数据的发现）：

  草案《draft-ietf-mboned-dorms》

  这定义了一种元数据被发现并传递到浏览器和网络的机制。（它使用 RESTCONF 和 YANG 模型。）其他 2 个（下面的 AMBI 和 CBACC）使用需要操作的信息来扩展该数据。

- AMBI（基于不对称清单的完整性）：

  草案《draft-itf-mboned-ambi》

这是浏览器如何验证接收到的数据的完整性并获取信息以监视数据包丢失的方式，而无需了解所传递协议的有效负载。它在经过身份验证的流中以带外方式发送数据包哈希，因此接收方知道它是否收到发送方发送的内容。

- CBACC（断路器辅助拥塞控制）：

  草案《draft-ietf-mboned-cbacc》

  这是浏览器（和网络）确保不超过容量限制的方式。这既定义了基于策略的限制，又使用 AMBI 提供的损失检测来响应损失的行为。每当有人超额订阅时，它就会注意到并切断多播流。

注意：可能还会有其他相关扩展。特别是，正在进行一些工作来定义 AMBI 的传输方式，该传输方式可以使用 ALTA 来验证数据包清单，从而与数据位于同一组播通道内。（我们希望这将对可以实现的延迟目标有所改进。）但是，此处列出的组件旨在提供足够的安全核心体系结构，并足以满足网络支持多播的可伸缩性问题的解决方案。启用，并且对于可以实现的延迟可接受的使用案例。

JavaScript API

这是一个加入 20 秒广播/组播流的示例。

```
// Multicast flow to join:

let multicastFlow = {
  source: '198.51.100.10',
  group: '232.10.10.1',
  port: 5001,
  dorms: 'dorms.example.com'
};

// Construct MulticastReceiver and subscribe to the multicast flow on the
// network:

let multicastReceiver=new MulticastReceiver(multicastFlow);

// Read multicast UDP packets:

let multicastReader=multicastReceiver.readable.getReader();
```

```
async function readMulticastData() {
  for(;;) {
    let { done, value } = await multicastReader.read();
    if(done) {
      return;
    } else {
      // value is an UInt8Array with the payload of one UDP packet.
      console.log("Got multicast packet with size "+value.length);
    }
  }
}

readMulticastData().then( () => {
  console.log("Cancel was called.");
}).catch( error => {
  console.log(error,"Error. Closecode is "+multicastReader.closecode);
});

// Cancel after 20 seconds:

setTimeout( () => {
  console.log("Canceling multicast");
  multicastReader.cancel();
},20000);
```

一旦 multicastReceiver 构建完成，浏览器将尝试加入网络上的多播流。每当有一个事件触发
a）read（）返回的承诺被拒绝，或
b）done 值变为 true 时，浏览器就会将多播流留在网络上。在这样的事件之后，不能再从多播接收器读取多播数据。因此，如果 JavaScript 希望在以后接收相同的或另一个多播流，它将需要构造一个新的 multicastReceiver。

如果 read（）返回的 promise 被拒绝，那么它将被拒绝，并带有一个信息性文本，该文本可能对调试有用，并记录到上面示例中的控制台中。除此之外，closecode 将被设置为几个众所周知的值之一，例如：

- MulticastReceiver.FLOW_PROBLEM：多播流本身存在问题。例如：
  - DORMS 无法识别给定源和组上的多播流。
  - 多播流具有比 DORMS 宣布的带宽更高的带宽。

    - 接收到组播数据包，但无法进行身份验证。
- MulticastReceiver.LOCAL_PROBLEM：本地设备出现问题，阻止了多播流的接收。例如：
    - 无法创建或关闭用于接收多播的套接字。
- MulticastReceiver.OVERSUBSCRIBED：达到了 CBACC 的超额订阅阈值（请参阅下文），因此离开了组播流或根本没有加入组播流。

获取 DORMS 元数据

考虑上一节中的示例对多播流的预订：

```
{
  source: '198.51.100.10',
  group: '232.10.10.1',
  port: 5001,
  dorms: 'dorms.example.com'
};
```

dorms.example.com 是一个域名和一个可选端口（默认端口为 443），浏览器将在该端口中尝试使用 HTTPS 来获取多播流的元数据，如 DORMS 规范中所述。除其他外，此元数据必须包括 AMBI 用来对多播流执行完整性检查的信任锚。下载元数据时，浏览器将验证服务器是否提供的有效证书 dorms.example.com。这确保了从 JavaScript 建立信任链，该 JavaScript 订阅了多播流以及浏览器使用 AMBI 执行的完整性检查。
我们还希望确保浏览器仅订阅要由浏览器订阅的多播流。这是为了防止例如：

- 恶意网站订阅了网络专用的多播流。
- 一个盗版网站，用于从普通网站订阅多播流。

有两种机制可以防止这种情况：

- 当使用 HTTPS 下载 DORMS 元数据时，浏览器将执行 CORS 检查。
- JavaScript 提供的 DORMS 服务器需要与 DORMS 规范中所述的 DORMS 反向 DNS 方案标识的相同（请参见下一段）。

第二个项目符号确保 JavaScript 只能从给定源 IP 订阅多播流，前提是该源 IP 的反向 DNS 已设置为允许它。更准确地说，在我们的示例中，当接收到多播订阅时，浏览器将对 SRV 域的记录进行反向 DNS 查找_dorms._tcp.10.100.51.198.in-addr.arpa。为了使订阅成功，此 SRV 记录必须提供主机名 dorms.example.com 和端口 443 的记录。如果在 JavaScript API 中也提供了其他端口，则可以使用该端口。显然，DNS 通常不被认为是安全的，因此在某些情况下，可能会入侵 DNS 来绕过此检查。此问题可能需要进一步考虑。

待定内容

TBD：在当前的 API 中，一次将一个数据包传递给 JavaScript。这类似于数据报在 WebTransport 中的工作方式。由于性能原因，我们可能希望一次传送多个数据包。我们目前正在 Chromium 中对此进行概要分析。

TBD：如果 JavaScript 可以访问 OS 或至少是浏览器从 OS 接收数据包的时间，那么可以使用精确的时间戳记就好了。这可以允许 JavaScript 实现协议，用于检测可用的带宽使用等机制的量 pathchirp 和/或 pathrate。我们要支持吗？如果这样做的话，API 应该是什么样？看起来，由 read()所返回的对象中的多余字段将是显而易见的地方。但这可能与 ReadableStream 系统不兼容。因此，我们可能需要将 UInt8Array 和数据包内容包装在一个额外的对象中？
TBD：为浏览器定义 CBACC 过度订阅阈值模型。最初的粗略想法是，以约 1mbps 的阈值开始，如果成功而没有损失，则增大阈值；如果损失持续升高，则减小阈值。

TBD：在基于 cbacc + ambi 的截止日期之前添加 api 扩展以用于丢失率报告。需要一些来自统计数据的预警。

TBD：我们可能希望扩展 API，以便浏览器可以通知 JavaScript CBACC 超额预订阈值即将达到。这可以允许 JavaScript 决定要取消订阅的多播流，而不是依靠浏览器选择一个或多个流并使用 MulticastReceiver.OVERSUBSCRIBEDclosecode 将其关闭。
TBD：如果 JavaScript 没有读取任何数据包，浏览器是否应该使用特定的 closecode 关闭多播流？与此相关的是，是否应该有一个 API，JavaScript 可以设置可在内部在 ReadableStream 中排队的最大数据包数？

TBD：浏览器是否应该自动从没有从网络接收到数据的多播流中取消订阅？答案很可能是“*否*”，因为没有数据的多播流订阅可用于向网络发送信号，表明它可能考虑创建该流。

待定：AMBI 可能会提供有关丢包的信息。我们是否希望该信息在 API 中可用？也许通过读取指示一个或多个丢弃的数据包的特殊数据包来进行。

待定：详细设计。这份早期草案旨在征询社区的反馈意见，即该通用方法是否因 IETF 草案中未提及的原因而注定了失败，或者一旦实现全部工作，是否可以尝试实施并填写所有细节。

§5 其它可替代设计

特定协议

有一个合理的问题，为什么我们不能通过为特定的传输协议（例如 FLUTE 或 NORM）定义 API 或通过添加对具有 RAMS 或其他 启用多播功能的 RTP 的支持来解决扩展问题 。

问题的一部分在于，许多现有应用程序（例如某些 ISP 当前部署的 IPTV 系统）不使用这些协议，即使某些部署使用私有协议且未公开提供，我们也希望不排除它们的用例。实现。

组播 IP 也是一个正在进行的研究领域，尚在开发新协议，例如 draft-pardue-quic-http-mcast。我们认为，启用与 Web 应用程序的这种试验将非常有价值。

此外，存在许多直接建立在不可靠的 UDP 传输之上的游戏或通信协议，这些协议特定于应用程序本身。这些应用程序通常也是专有的，或者至少是非标准的（但有时仍是已部署且有用的，并且可以从可伸缩性中受益）。

提出的体系结构的目的是提供所有必要的防护栏，以确保网络和接收器的安全性，以便可以像其他类型的流量一样安全地传送域间多播，同时允许进行实验以实现广泛的未开发潜力多种单向一对多协议设计。

WebTransport 网络运输工作组

已经提出，这可能适合 WebTransport 中的单向协议定义。

从目前的情况来看，该提议与“确保与 WebSockets 拥有相同的安全性”的 WebTransport 目标不兼容，因为该提议的一对多性质使其无法提供与 TLS 相同的安全性保证。

在与 WebTransport 的一些贡献者讨论之后，建议是将其作为独立的建议公开。

如果有足够的共识值得这样做，那么也许可以以某种方式使目标兼容并将该提案折叠到 WebTransport 的子集中。但是直到今天，这个想法已经被（非常临时地）考虑了，并且位于“被拒绝”和“推迟”之间。