将用户作为临时 CDN 节点，一种应用层的 DDoS 防御方法

First A. Author*. Second B. Author, Jr.**
Third C. Author***

**National Institute of Standards and Technology, Boulder, CO 80305 USA (Tel: 303-555-5555; e-mail: author@ boulder.nist.gov).*
***Colorado State University, Fort Collins, CO 80523 USA (e-mail: author@lamar. colostate.edu)*
**** Electrical Engineering Department, Seoul National University, Seoul, Korea, (e-mail: author@snu.ac.kr)}*

Abstract: 对于应用层的DDoS攻击，我们提出一种将用户设备作为临时CDN节点的防御方法；比如用户使用http/https协议访问网页时，服务器将网页或内容（如HTML、CSS、图片等）分块，每次只返回一个数据块给用户；用户要获取下一个数据块，必须完成一个任务：将已获得的数据块转发给其他N个用户；只有在其他N个用户确认收到数据块后，服务器才会返回给第一个用户下一个数据块；通过这种方式，可以减少对服务器的资源需求，同时增加攻击者的资源需求；

Keywords: Registration, Scan-matching, Lidar, Transformation.

1. INTRODUCTION

对于应用层的 DDoS 攻击，我们提出一种将用户设备作为临时 CDN 节点的防御方法；比如用户使用 http/https 协议访问网页时，服务器将网页或内容（如 HTML、CSS、图片等）分块，每次只返回一个数据块给用户；用户要获取下一个数据块，必须完成一个任务：将已获得的数据块转发给其他 N 个用户；只有在其他 N 个用户确认收到数据块后，服务器才会返回给第一个用户下一个数据块；通过这种方式，可以减少对服务器的资源需求，同时增加攻击者的资源需求；

2. DDoS 类型

DDoS（分布式拒绝服务）攻击是一种通过大量分布式设备向目标服务器、网络或服务发送海量流量或请求，从而导致其无法正常提供服务的攻击行为。根据攻击方式的不同，DDoS 攻击可以分为以下几种主要类型：

第一种 DDoS 攻击是基于流量的攻击（Volume-Based Attacks）；此类攻击通过大量的流量压垮目标的网络带宽或服务器资源，主要目的是耗尽带宽，常见类型如下；UDP 泛洪（UDP Flood）：攻击者向目标服务器发送大量的 UDP 数据包，目标需要花费资源去处理这些不必要的数据包，导致资源耗尽；ICMP 泛洪（ICMP Flood / Ping Flood）：攻击者发送大量的 ICMP 回显请求（ping 请求）给目标，消耗目标的带宽和计算能力；DNS 放大攻击（DNS Amplification Attack）：利用开放的 DNS 解析器，通过伪造源 IP 为目标服务器的 IP，发送小的请求数据包并诱使 DNS 服务器生成巨大的响应数据流量，将其发送到目标服务器；NTP 放大攻击（NTP Amplification Attack）：利用网络时间协议（NTP）中的 MONLIST 命令，将小请求放大为极大的响应流量并发送到目标；HTTP 泛洪（HTTP Flood）：攻击者发送大量的 HTTP 请求（例如 GET 或 POST 请求）到目标服务器，耗尽服务器的计算资源；

第二种 DDoS 攻击是协议攻击（Protocol Attacks）；此类攻击通过消耗服务器的协议处理能力或中间设备（如防火墙、负载均衡器）的资源，导致系统无法正常工作，常见类型如下；SYN 泛洪（SYN Flood）：利用 TCP 三次握手的机制，攻击者发送大量的 SYN 请求，但不完成后续的 ACK 响应，导致服务器资源耗尽；ACK 泛洪（ACK Flood）：攻击者发送大量伪造的 TCP ACK 数据包，消耗目标的资源；TCP 连接耗尽（TCP Connection Exhaustion）：攻击者建立大量的半开连接或长时间保持的连接，占用服务器的连接资源；Ping of Death：攻击者发送超大尺寸的 ICMP 数据包（超过标准 MTU 大小），导致目标系统崩溃或不可用；碎片化攻击（Fragmentation Attack）：攻击者发送故意分片错误的数据包（如 Teardrop 攻击），导致目标系统在重组数据包时崩溃；

第三种 DDoS 攻击是应用层攻击（Application Layer Attacks）；此类攻击针对特定的应用层协议或服务，利用其缺陷或设计特点耗尽系统资源，常见类型如下；HTTP Flood：攻击者发送大量合法的 HTTP 请求（例如 GET 或 POST 请求），通过模拟真实用户行为使攻击难以被检测；Slowloris：攻击者发送部分 HTTP 请求并保持连接不关闭，从而占用目标服务器的连接池资源，阻止其他合法用户访问；DNS 查询泛洪（DNS Query Flood）：攻击者发送大量合法的 DNS 查询请求，使目标 DNS 服务器无法响应其他用户的合法请求；SMTP 攻击：攻击者向邮件服务器发送大量伪造的邮件请求，导致邮件服务不可用；SSL/TLS 连接耗尽攻击：攻击者恶意发送大

量的 SSL/TLS 握手请求，消耗目标服务器的计算资源（因为 SSL/TLS 握手比普通连接更耗资源）；

3. 基于临时 CDN 节点的 DDoS 防御方法

3.1 算法步骤

对于应用层的 DDoS 攻击，我们提出一种将用户设备作为临时 CDN 节点的防御方法；比如用户使用 http/https 协议访问网页时，服务器将网页或内容（如 HTML、CSS、图片等）分块，每次只返回一个数据块给用户；用户要获取下一个数据块，必须完成一个任务：将已获得的数据块转发给其他 N 个用户；只有在其他 N 个用户确认收到数据块后，服务器才会返回给第一个用户下一个数据块；

设地址为ip_0的用户请求数据 a，该数据的大小为 size(a)；设每次发送的数据包的大小为 size(m)，那么数据 a 拆分的数据包个数为$k = ceil(size(a)/size(m))$；那么数据 A 得到的数据包集合为$A = \{a_1, \ldots, a_k\}$；我们计算每个数据包的哈希值集合为$H = \{h_1, \ldots, h_k\}$；此时服务器可以自行把数据包$a_1$发送给地址为$ip_0$的用户，也可以指挥其他已经得到数据包$a_1$的用户（比如$ip_{other}$）发送数据包$a_1$给地址为$ip_0$的用户（记为用户$ip_0$）；用户$ip_0$得到数据包$a_1$后，计算哈希值并发送会服务器，服务器检查哈希值是否正确，如果错误则需要重新按上述步骤再次发送；

用户ip_0得到数据包a_1后，服务器给用户ip_0发送需要转发的用户 IP 列表，设为$ip_{task} = \{ip_1, \ldots, ip_N\}$；这些用户收到数据包$a_1$后，会分别给服务器发送哈希值；如果用户$ip_0$完成了 N 个用户的转发任务，那么服务器开始发送下一个数据包a_2；按照相同的流程，逐步完成数据 A 的 k 个数据包的发送；

当发生 DDoS 攻击时，大量的用户并非真实的用户，而是攻击者控制的肉鸡；那么攻击者的某个肉鸡（用户ip_0）收到数据包a_1后，计算数据包a_1的哈希值h_1；然后服务器要求用户ip_0将数据包a_1转发给 N 个其他用户$ip_{task} = \{ip_1, \ldots, ip_N\}$；然而这里面大多数的用户都是攻击者控制的肉鸡，只有少数是真实用户；那么攻击者对于$ip_{task} = \{ip_1, \ldots, ip_N\}$中的肉鸡可以不必发送数据包$a_1$，而是直接发送哈希值$h_1$，省去了僵尸网络内部的网络消耗；对于$ip_{task} = \{ip_1, \ldots, ip_N\}$中的非肉鸡用户，攻击者仍然必须转发数据包$a_1$；

对于服务器来说，原本服务器需要将数据包a_1自行发送给所有的用户，现在只需要转发给少数的用户，剩余的让用户互相之间转发即可，服务器充当指挥；至于用户互相之间是否真实转发了数据包a_1（而不是转发了哈希值h_1），对服务器的消耗是没有关系的；对于攻击者来说，每次转发列表$ip_{task} = \{ip_1, \ldots, ip_N\}$中自己控制的肉鸡数量越多，那么需要真实转发数据包$a_1$的数量就越少；

然后我们讨论第二轮转发；现在$ip_{task} = \{ip_1, \ldots, ip_N\}$中所有用户都收到了数据包$a_1$（如果是肉鸡，可能只收到了哈希值$h_1$）；我们设$ip_{task}$中的某个用户为$ip_i$，该用户是攻击者控制的肉鸡，那么用户$ip_i$只收到了哈希值$h_1$；下面服务器给用户$ip_i$同样布置了转发 N 个用户的任务，完成这个任务才能收到数据包a_2；服务器给用户ip_i的任务列表为$ip_{task}^2 = \{ip_1^2, \ldots, ip_N^2\}$；其中$ip_{task}^2$中存在部分真实用户；那么用户$ip_i$就必须给这些真实用户发送数据包$a_1$；然而用户$ip_i$只收到了哈希值$h_1$，那么就没法完成这个任务；因此对于攻击者来说，要么只能转发所有的数据包，那么经过几轮转发，所有的肉鸡都逐渐没法完成任务，从而不能收到其他数据包；

在非攻击情况下，验证机制可以禁用或降低要求（如减少转发目标数量），以提高用户体验；在攻击情况下，服务器可以动态增加转发任务的难度（如增加 N 值），以对攻击者造成更大压力；

过程 1，服务器的伪代码

输入：接受到地址为ip_0的用户请求资源a（大小为 size(a)），每次发送的数据包大小size(m)，要求转发的用户列表ip_{task}

输出：数据包a发送给地址为ip_0的用户

将数据包a拆分为多个数据包，每个数据包的大小为 size(m)，得到数据包个数为$k = ceil(a/size(m))$，数据包的集合为$A = \{a_1, \ldots, a_k\}$；

计算数据包集合A的哈希值集合$H = \{h_1, \ldots, h_k\}$；

服务器向用户ip_0发送数据包的基本信息，每次发送的数据包大小size(m)，总共的数据包个数k；

for (数据包$a_i \in A$):

服务器发送数据包a_i给用户ip_0，或者指挥其他用户ip_{other}发送数据包a_i给用户ip_0；

等待用户ip_0发送数据包a_i的哈希值hi_{send}；

If($hi_{send} \neq h_i$):

If(数据包是服务器发送给用户ip_0):

重新发送数据包a_i；设定最大重发次数，超过则不再发送；

If(数据包是指挥其他用户ip_{other}发送给用户ip_0):

指挥其他用户ip_{other}重新发送；设定最大重发次数，超过则服务器亲自发送；再次超过最大重发次数，则不再发送；

服务器给用户ip_0发送任务列表$ip_{task}[i][N] = \{ip_1^i, \ldots, ip_N^i\}$，要求用户$ip_0$将数据包$a_i$发送给任何列表中的用户；

等待任何列表$ip_{task}[i][N]$中每个用户发送收到数据包a_i的哈希值$H = \{h_i^1, \ldots, h_i^N\}$；

for j in 1:N:

If($h_j \neq h_i, h_j \in H$):

要求用户ip_0重新发送数据包a_i给用户ip_j；如果超过最大重发次数，则更换新的任务；要求用户

ip_0将数据包发送给用户ip_{new}；

If(超过等待时间仍然没有收到任务列表$ip_{task}[i][N]$中每个用户的哈希值hi_{send}):

break

过程 2，客户端的伪代码

输入：服务器地址ip_{server}

输出：收到的数据包a

向服务器ip_{server}请求数据包a；

收到数据包a的基本信息，每次发送的数据包大小size(m)，总共的数据包个数k；

for i=1:k

接收数据包a_i，计算哈希值hi_{send}，将哈希值hi_{send}发送给服务器ip_{server}；

If(服务器回复数据包接收错误):

准备重新接收数据包a_i；如果超过最大重发次数，则接收失败；

接收任务列表$ip_{task}[i][N] = \{ip_1^i, \ldots, ip_N^i\}$，将数据包$a_i$发送给任务列表中的用户；

如果有任务发送失败，某些用户列表$ip_{task}[i][N]$中用户收到的数据包a_i计算的哈希值h_j和服务器的哈希值h_i不同；那么接收服务器要求的重发任务；

If(超过等待时间仍然没有收到服务器发下一个数据包a_{i+1}):

break

3.2 服务器的资源消耗

下面我们详细计算服务器的资源消耗；我们可以将服务器的资源消耗分为两类；第一类资源是所有用户相同的公共资源，如网站的首页、静态内容（CSS、JS、图片、视频等）；这些资源对所有用户是相同的，因此可以通过分布式传输将服务器的消耗降到最低；第二类资源是每个用户不同的个性化资源，如用户的动态请求（登录、增删查改操作、个性化数据等）；这些资源必须由服务器处理，无法通过用户之间的转发来减少资源消耗；

第一类资源的优化通过用户设备构建分布式 CDN 实现；极端情况下的理论最优解如下；理论上，服务器只需要发送一份公共资源内容；假设用户ip_0是第一个访问者，服务器将公共资源发送给用户ip_0；用户ip_0将内容转发给 N 个其他用户（用户$ip_1, \ldots, ip_N$）；接下来，这 N 个用户会分别转发给下一层的N_2个用户；通过这种链式分发，服务器的资源需求变为，服务器资源消耗 = 1 份公共资源 + 转发指令的元数据；这种处理方式的问题如下；如果完全依赖用户之间的分发，可能导致部分用户等待时间过长，尤其是当访问者数量激增时，初始几层的用户成为瓶颈；

为了减少用户等待时间，可以让服务器适当多发送几份公共资源，形成分布式分发网络的多个“起点”。这可以通过以下方式实现；首先我们可以动态调整主动发送份数；根据当前访问者的数量和分发速度，动态决定主动发送的份数；假设希望公共资源能在 3 层转发内覆盖所有用户，服务器可以主动发送足够的初始份数以加快传播；然后还可以采用多起点分发；服务器选取若干个“分发节点”（早期访问者），直接向他们发送完整内容；这些分发节点负责将内容传递到下一层用户；

在这种分布式模型下，服务器的消耗主要体现在以下方面；首先服务器的消耗是初始分发任务；服务器主动发送若干份公共资源作为分发网络的起点，并且消耗仅占用较少的带宽（理论上远小于直接向所有用户发送内容所需的带宽）；然后服务器的消耗是指挥和协调；服务器负责维护一个转发网络，指挥每个用户将内容转发给哪些目标用户，消耗主要是元数据开销（如转发指令、节点状态管理）；另外服务器的消耗是补充发送方面；如果某些用户未能及时收到内容，服务器可以主动补充发送；

对于第二类资源（用户个性化请求），由于这些请求必须由服务器处理，因此服务器的消耗无法通过分布式分发来减少；我们可以让访问者承担更多转发任务作为代价，作为一种平衡机制；

服务器总资源消耗分析如下；这里的服务器资源消耗包括发送数据的带宽资源和处理数据的 CPU 和内存等资源消耗；但是我们使用需要处理的数据来代替资源消耗，而不是具体的带宽、CPU 或内存的具体消耗；对于第一类资源的消耗，在理论最优情况下的消耗计算如下；

资源消耗（第一类资源）
= 1 份公共资源 + 转发指令的元数据

实际情况下，适当增加主动发送的份数（假设发送 M 份），服务器的资源消耗如下；

资源消耗(第一类资源)
= M × 公共资源 + 转发指令的元数据

对于第二类资源的消耗，服务器必须直接处理每个用户的动态请求，因此无法减少这部分消耗；假设用户数量为 U，每个用户动态请求的平均资源消耗为D_{req}，那么服务器资源消耗如下；

$$资源消耗(第二类资源) = U \times D_{req}$$

总的资源消耗为第一类资源消耗加第二类资源消耗；

3.3 攻击者的应对方法

这种 DDoS 防御方法在服务器、普通访问者和攻击者之间的策略类似博弈论中的非合作博弈；下面我们分析攻击者的应对策略和服务器的处理方式；

首先在每次获得数据包a_i后，需要完成转发任务$ip_{task} = \{ip_1.,\ldots,ip_N\}$才能获得下一个数据包$a_{i+1}$；如果任务列表$ip_{task}$中存在攻击者控制的肉鸡$ip_i$，那么攻击者可以不用发送数据包$a_i$，而是发送数据包的哈希值$hi_{send}$；然后下一轮转发中，肉鸡$ip_i$得到的转发任务$ip^2_{task} = \{ip^2_1,\ldots,ip^2_N\}$中，可能存在非肉鸡的普通用户；那么此时肉鸡$ip_i$必须重新获得数据包$a_i$，然后将数据包$a_i$发送给这个普通用户；否则肉鸡$ip_i$不能获得下一个数据包$a_{i+1}$；这样可以增加攻击者的转发成本，或者经过几轮转发，很多肉鸡没法完成转发任务，从而退出 DDoS 攻击；

然后攻击者可能不执行转发任务，而是只获取第一个数据包a_1，然后断开和服务器的连接；然后重新向服务器请求数据包a_1；那么我们可以限制前几个数据包的大小，或者逐步增加每次发送的数据包；此外，这种方法可以迫使攻击者从应用层的 DDoS 攻击趋向于基于流量的 DDoS 攻击；然后这种基于流量的 DDoS 攻击具有较为明显的特征（比如同个 IP 短时间大量连接），从而可以识别并屏蔽这些用户；

此外还存在如下的攻击形式；对于正常访问者ip_{normal}，在获得数据包a_i后，同样需要完成转发任务$ip_{task} = \{ip_1,\ldots,ip_N\}$；然后转发任务$ip_{task}$中可能存在很多攻击者控制的肉鸡；那么这些肉鸡即使收到了正常访问者ip_{normal}发送的数据包a_i，照样不给服务器发送数据包的哈希值hi_{send}（当作没有收到）；这样可以阻碍正常访问者ip_{normal}完成转发任务，从而使得正常访问者ip_{normal}没法获得下一个数据包a_{i+1}；

对此我们可以让正常访问者ip_{normal}在超过最大重发次数后，转向完成 PoW（Prove of Work）任务，完成这个 PoW 任务后，同样可以获得下一个数据包a_{i+1}；PoW 任务是要求访问者执行一个具有计算量的任务，完成后可以继续访问；比如 Google 的部分验证网页要求访问者执行一段 Javascript 代码，完成后才能继续访问；上述处理方式的特点如下；首先即使攻击者的肉鸡不配合，正常访问者ip_{normal}仍然可以通过完成 PoW 任务获得下一个数据包a_{i+1}；然后对于攻击者控制的肉鸡ip_i，如果多次没有接收到数据包a_i，可能被服务器拒绝继续发送数据包，从而将肉鸡ip_i排除掉；

4. 基于流量的 DDoS 攻击

除了应用层的 DDoS 攻击，还有基于流量的 DDoS 攻击；此类攻击通过大量的流量压垮目标的网络带宽或服务器资源，主要目的是耗尽带宽，常见类型如下；UDP 泛洪（UDP Flood）：攻击者向目标服务器发送大量的 UDP 数据包，目标需要花费资源去处理这些不必要的数据包，导致资源耗尽；ICMP 泛洪（ICMP Flood / Ping Flood）：攻击者发送大量的 ICMP 回显请求（ping 请求）给目标，消耗目标的带宽和计算能力；DNS 放大攻击（DNS Amplification Attack）：利用开放的 DNS 解析器，通过伪造源 IP 为目标服务器的 IP，发送小的请求数据包并诱使 DNS 服务器生成巨大的响应数据流量，将其发送到目标服务器；NTP 放大攻击（NTP Amplification Attack）：利用网络时间协议（NTP）中的 MONLIST 命令，将小请求放大为极大的响应流量并发送到目标；HTTP 泛洪（HTTP Flood）：攻击者发送大量的 HTTP 请求（例如 GET 或 POST 请求）到目标服务器，耗尽服务器的计算资源；

对于基于流量的 DDoS 攻击中，网站提供者的损失包括带宽费用的损失和服务器计算资源的损失；我们认为其中网站提供者的带宽费用的损失一定程度上是网络运行商的带宽收费制度问题造成的；有些网络运行商对网站提供者的上行带宽和下行带宽都需要收费；发生 DDoS 攻击时，网站提供者会出现大量的下行带宽（服务的请求数据），这些带宽并不是网站提供者可以决定的，而是攻击者发出的；那么即使网站提供者不对任何攻击请求做出响应或者保持服务器关机，仍然需要支付这部分下行带宽的费用；虽然在实际应用中，网站提供者可以选择固定速度的带宽或者将攻击流量引到黑洞路由的方式限制下行带宽的费用；因此我们认为更加合理的网络运行商的带宽收费方式应该是采用上行带宽收费，上行带宽是网站提供者主动发出的；这个类似拨打电话时，电话拨出者需要为电话付费；事实上有些网站运营商已经采用这种只对上行带宽收费的方式；

然后如果对网站提供者和普通用户都采用上行带宽收费，会造成如下的问题；普通用户在访问网站时，上行带宽非常少（只有请求数据）；普通用户的大部分网络使用都是下行带宽（下载网站提供的数据）；这样会造成大部分的网络运营商的收费都需要网站提供者支付；那么网站提供者会仔细权衡某个用户的请求是否值得响应，从而提高网络上信息传递的门槛；

因此为了平衡网站提供者和普通用户的网络费用支出，对于普通用户可以采用上行带宽和下行带宽同时收费；或者普通用户只对下行带宽收费，但是限制上行带宽的速率，防止普通用户滥用上行带宽，对网站提供者造成 DDoS 攻击的效果；这也是当前多数网络运营商的收费方式；

REFERENCES

Brown, F., Harris, M.G., and Other, A.N. (1998). Name of paper. In Name(s) of editor(s) (ed.), *Name of book in italics*, page numbers. Publisher, Place of publication.

Smith, S.E. (2004). *Name of book in italics*, page or chapter numbers if relevant. Publisher, Place of publication.

Smith, S.E. and Jones, L.Q. (2008). Name of paper. *Name of journal in italics*, volume (number), page numbers.

【待补充，调整格式后加入】