

# 基于 eBPF 实现 Linux 防火墙

黎浩然

## 摘要

随着互联网的 Linux 服务器中容器化应用程序的大量增加以及网络带宽的不断提升, Linux 事实标准的防火墙——iptables, 在性能以及可拓展性方面逐渐难以适应当前的需求。eBPF 技术的兴起为 Linux 的防火墙提供了另一种可行的解决方案。本课题的研究内容是基于 eBPF 提供的接口, 遵守传统 iptables 的语法和语义, 设计并实现 bpf-iptables。实验数据表明 bpf-iptables 比传统的 iptables 有非常显著的性能提升。本人的主要工作则是在完全理解 bpf-iptables 框架的基础上, 对其改进并通过实验验证改进后的 bpf-iptables 性能有所提升。

**关键词:** eBPF; iptables

## 1 引言

Iptables 是运行在用户空间的应用软件, 通过控制 Linux 内核 netfilter 模块, 来管理网络数据包的处理和转发。自从 Linux 2.4.0 引入以来至今, iptables 仍然是 Linux 主机防火墙系统中最核心的安全组件。Linux 3.13 版本以后, nftables 取代之前的 iptables, 但仍然提供 iptables 命令作为兼容接口。iptables 允许对流量应用不同的策略, 例如可用于防止潜在的网络攻击或者限制不同机器之间特定类型的数据包。但是 iptables 的线性搜索算法限制了它的策略数量及处理速度, 它的语法并不总是直观以及使用了不少难以理解和维护的旧代码库。随着网络中 Linux 服务器中容器化程序的大量增加以及带宽的不断提升, 当前的 iptables 可能无法再满足性能、灵活性和可拓展性等方面的需求<sup>[1]</sup>。

eBPF 技术的出现使得系统管理员可以注入任何不会影响主机安全运行的代码到 Linux 内核中<sup>[2]</sup>。eBPF 程序可以被加载到在网络协议栈中不同的挂钩点, 例如 eXpress Data Path(XDP)<sup>[3]</sup>和 Traffic Control(TC), 从而拦截网络数据包, 并且选择放行或丢弃这些数据包。Daniel Borkmann 提出的 bpfILTER 框架<sup>[4]</sup>可以将现有的 iptables 规则透明地转换为 eBPF 程序。系统管理员可以继续使用现有的基于 iptables 的规则配置方法, 而无需关心数据包的过滤是否使用 eBPF 程序实现的。bpfILTER 引入了一种称为用户模式助手 (User Mode Helper) 的内核模块, 可以将功能委托给用户空间进程实现, 以此将在用户空间创建的 eBPF 程序注入内核中。目前, bpfILTER 框架的工作主要集中在将 iptables 规则转换为 eBPF 指令的翻译架构的设计, 并通过对相关概念的一个小验证来表明在内核中、甚至在物理硬件中过滤数据包的优势<sup>[5]</sup>。

本课题的工作沿着 bpfILTER 提议进行, 也就是创建一个更快、更具可拓展性的 iptables 的替代品, 但是还有以下两个额外的挑战: 首先是需要保留 iptables 的过滤语义, 并且提供透明的转换方案, 使网络管理员不会注意到转换前后的任何差异。这意味着不仅需要遵守 iptables 的语法, 还需要准确地实现其行为。细微的差异可能会给那些依赖 iptables 来保护的系统带来严重的安全隐患。其次是提高

iptables 的速度和可拓展性。iptables 的线性搜索算法是限制其可拓展性的主要原因，也是其性能受限的重要原因。

基于以上的考虑，本课题提出了基于 eBPF 的 Linux 防火墙 bpf-iptables，在 eBPF 中实现了另一种数据包过滤架构。bpf-iptables 在保持相同于 iptables 的过滤语义的同时，大幅改进其性能和可拓展性。bpf-iptables 充分利用了 Linux 内核的接口来提高处理数据包的吞吐量。其中一个重要接口就是 XDP，它用于为不需要被 Linux 网络栈处理的数据包提供了一条快速通道。例如当前收到的数据包的目的主机不是本机时，XDP 处的 eBPF 程序根据过滤表决定直接将该数据包递送到指定端口或者立即丢弃。当前的 bpf-iptables 只实现了 iptables 和 nftables 的 FILTER 表，RAW、NAT 和 MANGLE 表留给未来的工作中实现。

## 2 挑战与假设

### 2.1 与 iptables 保持语义一致性

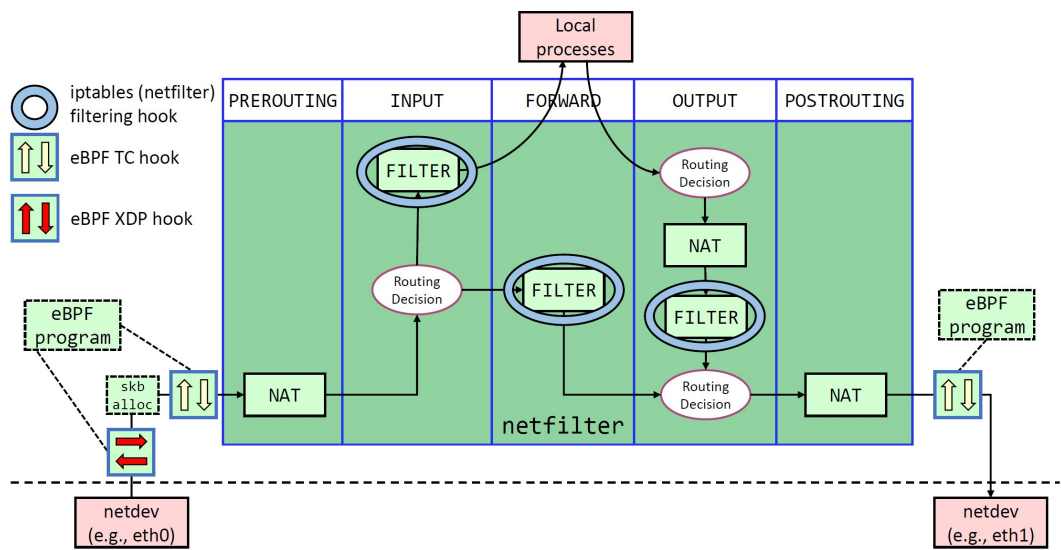


图 1: netfilter 和 eBPF 挂钩子的位置

Iptables 和 bpf-iptables 之间的主要区别在于两者的底层框架分别是 netfilter 和 eBPF。Iptables 定义三个默认的规则链，用于在 Linux 网络栈中不同的位置过滤数据包。如图 1 所示，INPUT、FORWARD 和 OUTPUT 三个规则链分别对应 netfilter 的三个挂钩点。此外，eBPF 程序可以附加到不同的挂钩点。如图 1 所示，入口流量可以在 XDP 或者 TC 模块中被截获，因此截获的时机要早于 netfilter 的 INPUT 链；出口流量只能在 TC 模块中被截获，截获的时机要晚于 netfilter 的 OUTPUT 链。这种挂钩点的差异会导致一些问题：例如规则 “iptables -A INPUT -j DROP” 会让 iptables 丢弃所有指向本主机的流量，但是并不会影响本主机需要转发的流量。如果在 XDP 或 TC 上直接应用类似全部丢弃的规则，将丢弃所有的入口流量，包括目的主机并不是本机的流量。因此，bpf-iptables 要保持与 iptables 的语义一致性，必须提前确认这些传入数据包将进入 INPUT 还是 FORWARD 链上。

### 2.2 在 eBPF 中实现高效匹配算法

由于 eBPF 环境的内在限制，选择并实现更好的规则匹配算法具有很大的挑战性<sup>[6]</sup>。事实上，诸如向量叉积<sup>[7]</sup>、决策树方法<sup>[8]</sup>这些更优秀的匹配算法依赖于复杂的数据结构或者具有不可预计的内存开销上限，在 eBPF 环境难以实现。因此，bpf-iptables 所选择的匹配算法不仅必须是高效和可拓展的，还需要在当前的 eBPF 技术下可行。

### 2.3 支持有状态的过滤 (conntrack)

Netfilter 可以跟踪 ICMP/TCP/UDP 的会话状态并将其存储在会话表 (conntrack) 中。Iptables 可以利用此表来支持基于会话状态接受或丢弃数据包的规则。例如，iptables 可以只接受属于 NEW 或者 ESTABLISHED 的 TCP 数据包。如图 1 所示，由于 bpf-iptables 在数据包进入 netfilter 之前过滤数据包，bpf-iptables 无法使用 Linux 的 conntrack 模块对数据包进行基于会话状态的分类。为此 bpf-iptables 必须自己实现等效的组件。

## 3 总体架构

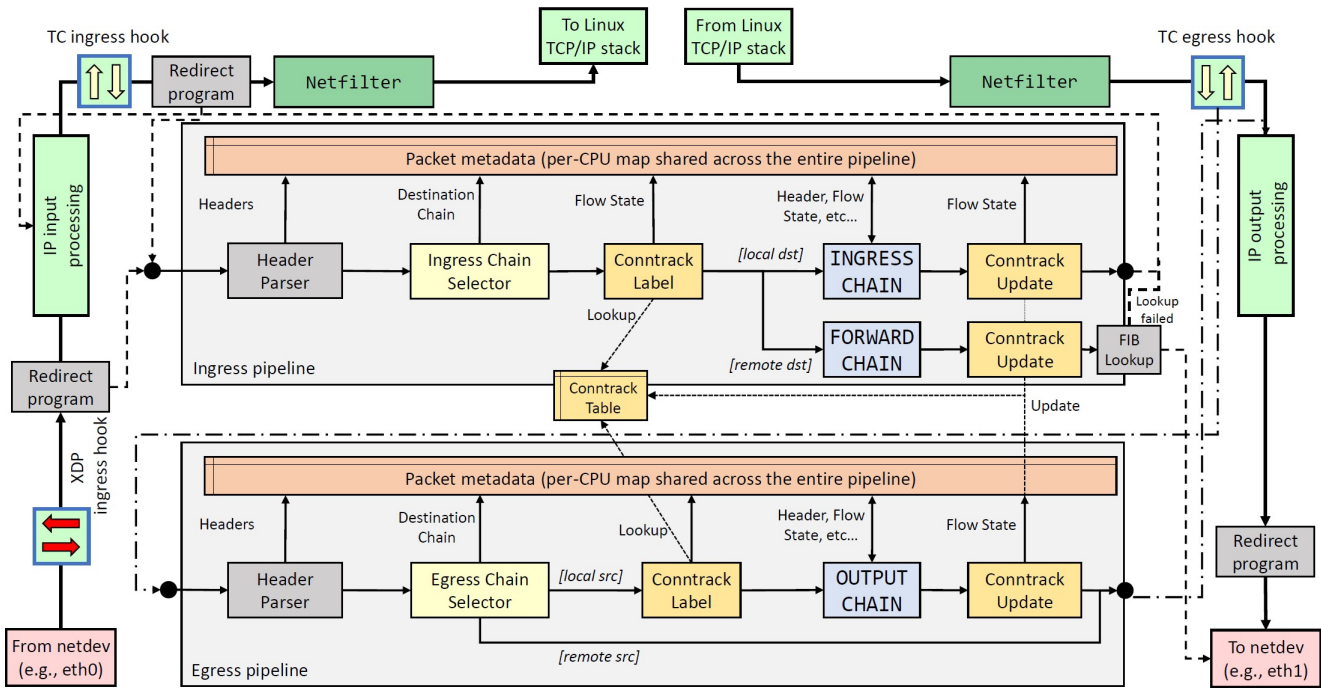


图 2: bpf-iptables 的总体架构

图 2 展示了 bpf-iptables 的整体系统架构。数据平面包括四类主要的 eBPF 程序：蓝色组的程序实现了 INPUT、FORWARD 和 OUTPUT 三条链上过滤行为；黄色组的程序实现了保留 iptables 语义所需的逻辑；橙色组的程序用于会话状态 (conntrack) 的跟踪、匹配与更新；灰色组的程序用于一些辅助任务，如数据包字段的解析。

附加在 XDP 或者 TC 模块上的 eBPF 程序截获到进入主机的数据包时会调用 ingress pipeline。bpf-iptables 默认在 XDP 模式下工作，但是这需要网络接口卡的驱动程序支持 XDP 模式；当网络接口卡不兼容 XDP 时，bpf-iptables 会自动回退到 TC 模式。由于 XDP 模式在网卡出口不可用，所以数据包离开主机时只能调用附加在 TC 模块上程序进入 egress pipeline，

进入 ingress pipeline 之后，数据包会根据路由决策决定进入 INPUT 链还是 FORWARD 链；当进入 INPUT 链上，如果数据包没有决定被丢弃，它将通过主机的网络协议栈最终到达应用程序。当进入 FORWARD 链上，无论数据包被接受还是丢弃，都不会在经过网络协议栈。如果该数据包需要被转发，则会直接通过 FIB Lookup 模块递送到 OUTPUT 链上处理。

控制平面（图 2 中并未展示）在用户空间中执行并提供三个主要功能：(i) bpf-iptables 数据平面的初始化与更新, (ii) 初始化运行匹配算法所需的 eBPF 数据结构, (iii) 监视网络接口卡的数量及状态的变化，这是完全模拟 iptables 行为所必需的，即需要处理来自所有网络接口的流量。

4 数据平面

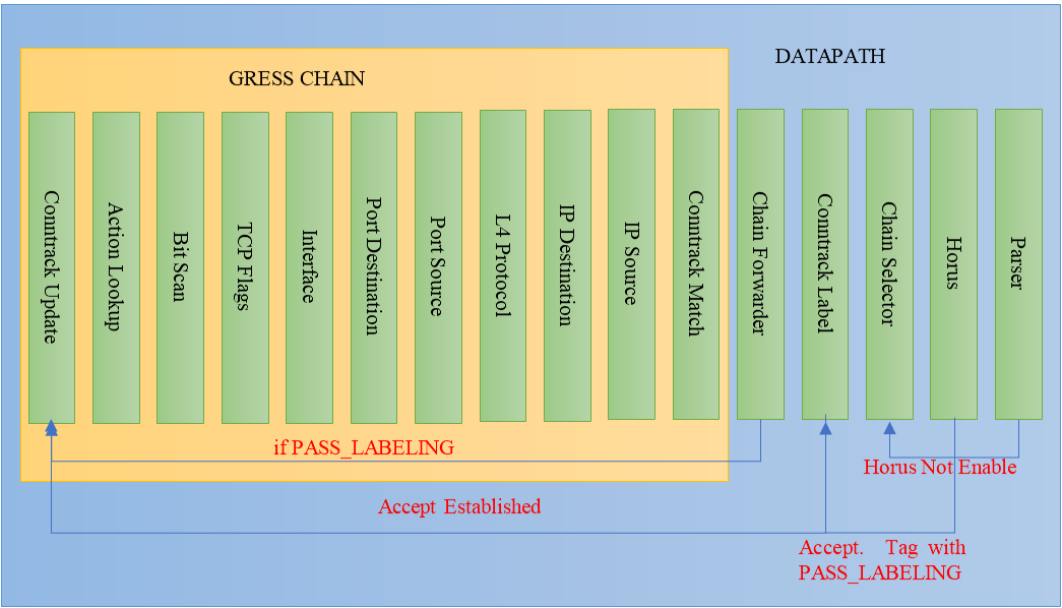


图 3: bpf-iptables 的数据平面

图 3展示了 bpf-iptables 数据平面的所有 16 个模块，其中每个链 (CHAIN) 实例 (INPUT、FORWARD 和 OUTPUT) 包含 10 个模块。红色方框里面展示了这些模块的调用顺序：由上至下是模块的默认调用顺序，红色曲线箭头表示其中一些符合特定规则的数据包可能会跳过其中一些模块。接下来分别对各个模块的作用以及实现作介绍，在此之前，我先来介绍各条链中的规则匹配算法。

4.1 链中的匹配算法

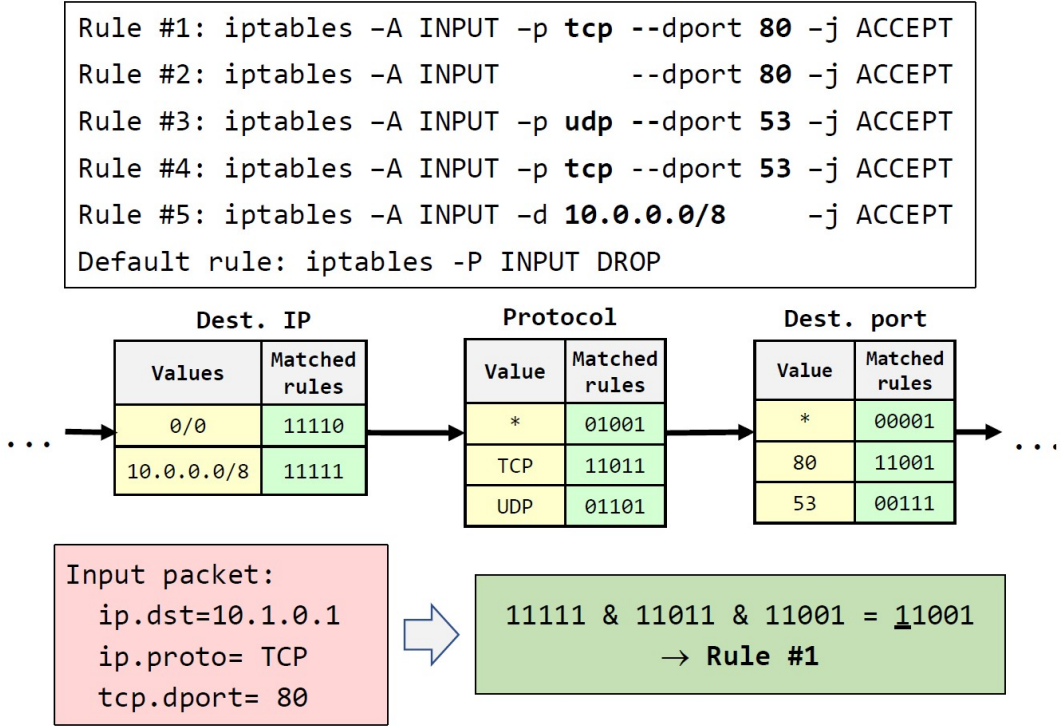


图 4: Linear Bit-Vector Search

为了克服 iptables 线性搜索算法带来的性能缺失，bpf-iptables 采用了更高效的线性比特向量搜索 (Linear Bit-Vector Search, LBVS)<sup>[9]</sup>算法。LBVS 算法遵循分治法的思想，其内在的流水线结构能完美地与当前的 eBPF 技术相契合。LBVS 算法首先为过滤规则集中出现的每个字段构造一个列表，例如图

4展示的过滤规则集总共出现的3个字段对应的3个列表。在每个列表中，`values` 初始化为该规则字段在过滤规则集中出现过的所有实例加上一个通配值，`match rules` 为一个比特序列，该比特序列的每个比特分别对应于过滤规则集中的每条规则，因此该比特序列长度就是过滤规则的总数量。当 `values` 字段的实例匹配与某条过滤规则时，就将 `match rules` 字段中对应于那条过滤规则的比特置为“1”，否则置为“0”。因此，LBVS 算法的运行过程如图 4所示，将每个需要匹配的数据包字段匹配的规则对应的比特向量进行“与”运算，最终得到的比特向量表示该数据包匹配的规则。由于默认在 `iptables` 的过滤规则集前面的规则具有更高的优先级，因此最终取过滤规则集中第一条对应比特为“1”的规则为该数据包最终匹配规则，也就是图 4中的第一条规则。

## 4.2 Parser

如图 2所示，`bpf-iptables` 的 `ingress pipeline` 和 `egress pipeline` 都从 `Parser` 模块开始，该模块的主要作用是提取当前过滤规则所需的数据包字段，将每个字段值存储在由 `pipeline` 中所有 `eBPF` 程序共享的 `per-CPU map` 中，即图 2中的数据包元数据 (`packet metadata`)。这样就避免后续 `eBPF` 程序重复解析数据包头部的工作，保证了更好的性能和更紧凑的处理代码。`Parser` 的代码是动态生成的，一旦过滤规则的更新导致 `Parser` 需要提取的字段发生变化，控制平面会根据新的过滤规则重新生成、编译新的 `eBPF` 程序并注入到内核中。因此，`Parser` 的处理代价主要是由当前过滤规则集需要解析的字段的数量决定的，而不像线性搜索算法那样主要由规则集的数量决定。

## 4.3 HOMogeneous Ruleset analySis (HORUS)

由网络管理软件自动配置的防火墙通常会在过滤规则集的开头插入一组在包含相同字段的同构规则，称为 `HORUS` 规则集。这些具有更高优先级的规则可以被提前到在 `Parser` 与 `Chain Selector` 之间匹配数据包。当此功能被激活时，`bpf-iptables` 会在 `Parser` 与 `Chain Selector` 之间插入 `HORUS` 模块。数据包经过 `HORUS` 模块时，`bpf-iptables` 会先尝试将该数据包与 `HORUS` 规则匹配。如果匹配到规则并且动作为丢弃时该数据包会被立即丢弃；否则如果匹配到规则并且动作为接受时，该数据包会先经过 `Conntrack Label` 以更新 `Conntrack` 相关的标记，然后由 `Chain Forwarder` 直接递交给 `Conntrack Update` 模块，不会再经过 `GRESS CHAIN` 中的模块；否则，也就是该数据包没有匹配到任何 `HORUS` 规则，该数据包会正常进入 `Conntrack Label` 及后续的模块，使用 `LBVS` 算法匹配剩余的规则。

`HORUS` 集中的规则必须满足以下些条件：1) 必须保持和原规则集一致的先后顺序，例如在 `HORUS` 规则集中规则 A 出现在规则 B 的前面，那么原规则集中必须是规则 A 出现在规则 B 的前面；2) 任何规则不能与不在 `HORUS` 集中的其它具有更高的优先级的规则相冲突，例如原规则集中规则 A 出现在规则 B 的前面，但 `HORUS` 规则集中只包含规则 B 没有包含规则 A，那么所有能够匹配规则 B 的数据包不应该能够匹配规则 A。也就是说，`HORUS` 不应该改变 `iptables` 的任何预期行为。

`HORUS` 发挥巨大优势的一个重要场景是在 `DoS` 攻击下。事实上，如果 `HORUS` 规则集中的所有规则都包含一个 `DROP` 动作，匹配的数据包将被立即丢弃，因此利用 `XDP` 提供的早期处理能力允许以高速率丢弃数据包和在支持卸载简单 `eBPF` 程序的硬件加速器上运行该程序能力，可以进一步降低系统负载和资源消耗。

然而，`HORUS` 默认是不启用的，因为一般情况下它只能包含很少一部分规则，对 `bpf-iptables` 的整体性能影响很小。在第六章中我将讨论对 `HORUS` 规则集选取的改进。



## 4.4 Chain Selector and Forward

Chain Selector 主要有两个作用。一个作用是对于进入 ingress pipeline 的数据包，Chain Selector 通过查询存储在 map 中的 IP 来判断数据包的目标地址是否为本机的 IP 地址，然后据此来决定数据包应该递交到 INPUT 链还是 FORWARD 链；对于进入 egress pipeline 的流量，Chain Selector 根据数据包的源 IP 是否为本机的 IP 地址判断该数据包是来自本机的 FORWARD 链还是由本机直接发出，因为经过 FORWARD 链的数据包已经在 pipeline 中处理过，不应再由 OUTPUT 链处理。此外，Chain Selector 优化了一种非常常见的情况：当 INPUT 链的第一条规则是接受所有 conntrack 为 ESTABLISHED 的数据包时，匹配此规则的数据包无需进一步处理直接递交给操作系统协议栈。Chain Forwarder 则是 Chain Selector 的动作执行器，负责执行 Chain Selector 的决策。

## 4.5 Conntrack

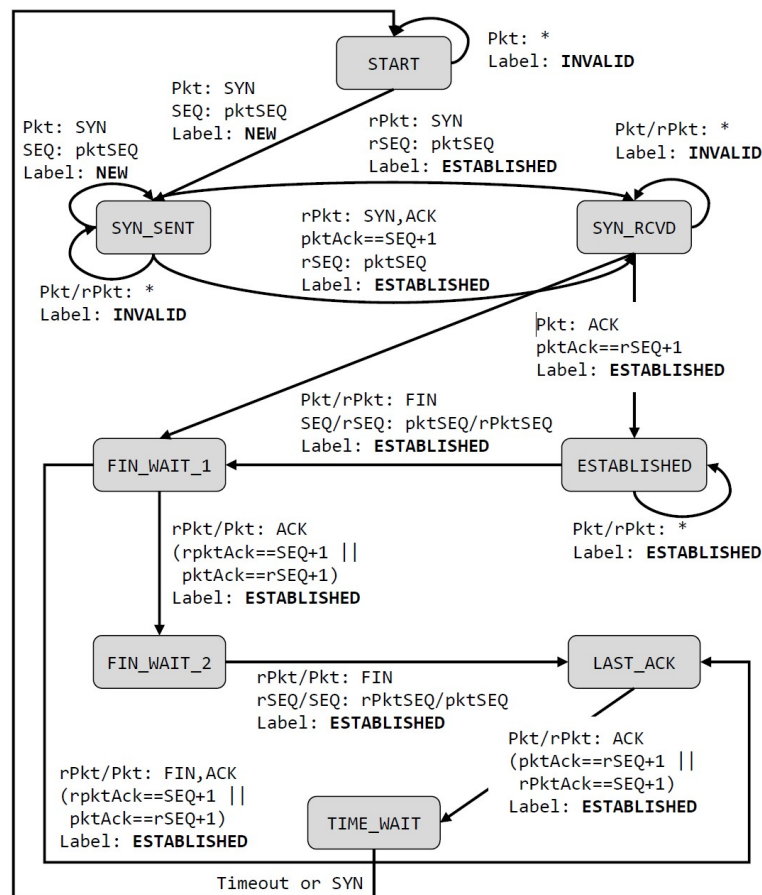


图 5: bpf-iptables conntrack 模块中的 TCP 状态机。灰色框表示 conntrack 表中保存的状态；Labels 代表数据包进入 INPUT、FORWARD 和 OUTPUT 链前第一个 conntrack 模块分配的值。

Conntrack(Connection Tracking) 是 iptables 中的概念。为了支持基于状态的过滤, bpf-iptables 实现了自己的连接跟踪模块, 其特点是在 ingress pipeline 和 egress pipeline 中放置了五个额外的 eBPF 程序, 根据当前流量所属连接的状态过滤数据包。这些模块共享相同的 BPF\_HASH conntrack map, 如图 2。同时, 为了正确更新连接状态, bpf-iptables conntrack 必须拦截双向流量(即主机到 Internet 和 Internet 到主机的流量)。即使用户仅在 INPUT 链上配置过滤规则, 传出数据包都必须由位于 egress pipeline 中的 conntrack 模块处理。bpf-iptables 连接跟踪支持 TCP、UDP 和 ICMP 流量, 尽管它不处理相关连接等高级功能, 也不支持 IP 重组。

## 4.6 Bit Scan

由于每个匹配规则在位向量中都表示为“1”，bpf-iptables 使用 N 个 64 位无符号整数数组来支持大量规则。因此，在执行按位与时，当前的 eBPF 程序必须对整个数组执行 N 次循环才能比较两个位向量。鉴于 eBPF 上没有循环，这个过程需要循环展开，因此受到 eBPF 程序中可能指令的最大数量的限制，因此也限制了支持规则的最大数量。

## 4.7 Action Lookup

数据包一旦到达 pipeline 的末端，最后一个程序必须找到与当前数据包匹配的规则。该程序从共享 map 中提取位向量并在该在位向量中查找第一个位为 1 的位置。然后，该程序使用该位置从给定的 BPF\_ARRAY 映射中检索与该规则关联的最终操作并且最终应用该操作。显然，如果没有匹配的规则，则应用默认操作。

# 5 控制平面

## 5.1 启动阶段

为了像 iptables 一样工作，bpf-iptables 必须拦截所有传入和传出的流量，并在其自定义 eBPF pipeline 中处理这些数据包。启动时，bpf-iptables 将一个小的 eBPF 重定向程序附加到从根命名空间可见的每个网卡的 ingress 和 egress 挂钩，如图 2 所示。该程序拦截所有通过接口的数据包并调用 ingress/egress pipeline 的第一个程序。使得能够创建处理来自所有网卡的数据包的单个处理 pipeline，因为 eBPF 程序无法跨越不同的网卡相互调用。最后，bpf-iptables 检索任何网卡上活动的所有本地 IP 地址并交由在 Chain Selector 中配置；这个初始化阶段是通过订阅适当的 Netlink 事件集来完成的。

## 5.2 Netlink 事件通知

每当收到新的 Netlink 通知时，bpf-iptables 都会检查其是否与根命名空间中的特定事件相关，例如接口的创建或 IP 地址的更新。在第一种情况下，重定向程序附加到新接口的 eBPF 挂钩，使 bpf-iptables 能够检查其流量。在第二种情况下，bpf-iptables 使用新地址更新 Chain Selector 中使用的本地 IP 列表。

## 5.3 规则集变更

当用户更新规则集时，bpf-iptables 首先平行创建新的链以及对应的 eBPF 程序，接着开始执行预处理算法计算每个字段的值——位向量键值对，然后并这些键值对插入到该链的 eBPF map 中。这个阶段的最后一步是将生成的值插入到它们的 eBPF map 中。每个匹配字段都有一个默认 map；但是，bpf-iptables 还可以根据当前规则集值在运行时选择映射类型。例如，LPM\_TRIE 用作 IP 地址的默认 map；但是，如果当前规则集仅包含具有固定 IP 地址的规则，它会将映射更改为 HASH\_TABLE，从而使匹配更加高效。在实例化 pipeline 之前，bpf-iptables 通过重新生成和重新编译最能代表当前规则集的 eBPF 程序来修改每个模块的行为。当为给定字段选择了最合适的映射时，bpf-iptables 会用计算出的值——位向量对填充它。eBPF map 和字段类型的组合会影响 bpf-iptables 表示通配符规则的方式。对于用于匹配 IP 地址的 LPM\_TRIE 等映射，通配符可以表示为值 0.0.0.0/0，它可以作为任何其他值插入。另一方面，对于使用 HASH\_MAP 的 L4 源端口和目标端口，bpf-iptables 将通配符值实例化为 eBPF 程序中硬编码的变量；当表中的匹配失败时，它将使用直接从 map 中检索到的通配符变量。

Bpf-iptables 对具有有限数量可能值的字段采用了先前算法的变体，其中不是为字段生成不同值

的集合，而是为该值生成所有可能的组合。优点是：(1) 它不需要为通配符生成单独的位向量，因为所有可能的组合都已经包含在 `map` 中；(2) 可以使用 `eBPF ARRAY_MAP` 实现，与其他 `map` 相比，它更快。一个例子是处理 `TCP` 标志；由于该字段的所有可能值的数量是有限的，因此用所有可能的情况扩展整个字段而不是精确计算使用中的值会更有效。

## 6 复现细节

### 6.1 与已有开源代码对比

#### 6.1.1 HORUS 集选取规则的改进

在大部分场景下，规则的字段数量并不是都一致的。`bpf-iptables` 的 `HORUS` 规则集选取的是位于原过滤规则集的开头的具有相同字段的过滤规则，因此会导致 `HORUS` 规则集往往很小。我对此作了一些改进，使得 `HORUS` 规则集在某些场景下不至于有过少的规则。图 6 是改进后的 `HORUS` 规则集选取的例子，较改进前增加了更多被选中的规则。

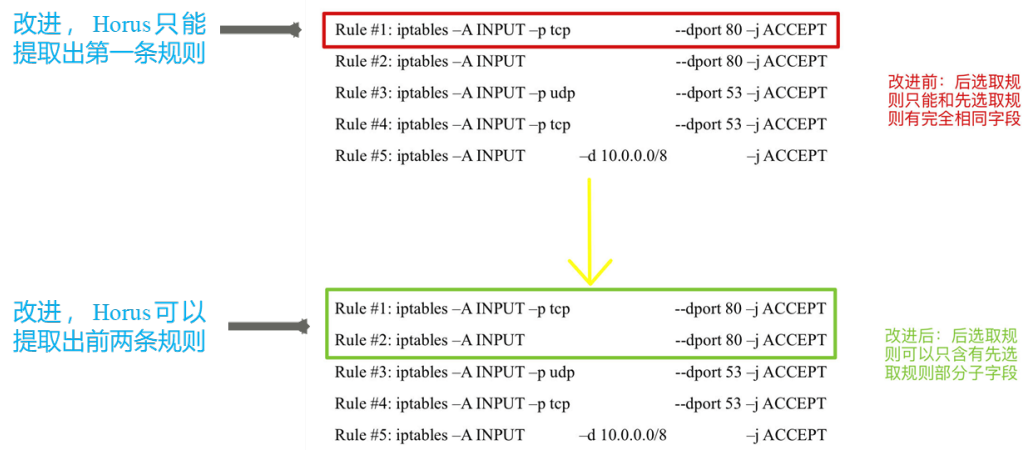


图 6: HORUS 改进后示例

#### 6.1.2 自适应的字段比较次序

字段的比较次序也会影响 `bpf-iptables` 的性能。例如，当比较到某一字段的中间结果位向量为全零时，后续的字段比较就没有意义，因为此时可以断定没有任何规则可以匹配此数据包，应该触发 `early break`。为了改进这个问题，有必要示具体的情况对字段的比较先后排序，使得更多根本不会匹配任何过滤规则的数据包能更快地触发 `early break`。通过观测发现，一个字段在所有过滤规则里面出现次数越多，该字段触发 `early break` 的概率越高。因此当每次过滤规则集更新时，`bpf-iptables-mod` 据此来对负责各个字段比较的 `eBPF` 程序排序。

图 7 展示了改进的 `LBVS` 算法受益的一个例子。可以看到，由于目标端口字段要比目标 `IP` 字段在原始规则集中的出现频率更高，因此应该先比较目标端口字段，再比较协议字段，最后比较目标 `IP` 字段。图 7 中所示的 `Input` 数据包可以在第二个字段比较之后触发 `early break`，而不必像改进前那样比较完每一个字段。当然，值得注意的是并不是表示字段出现的频率越高放在前面就越好。这是因为当一个字段在某条过滤规则中出现时，数据包必须对应于此字段的实例才可能匹配此规则；否则如果该字段没有出现，则任何满足其它字段的数据包都可以匹配此规则。



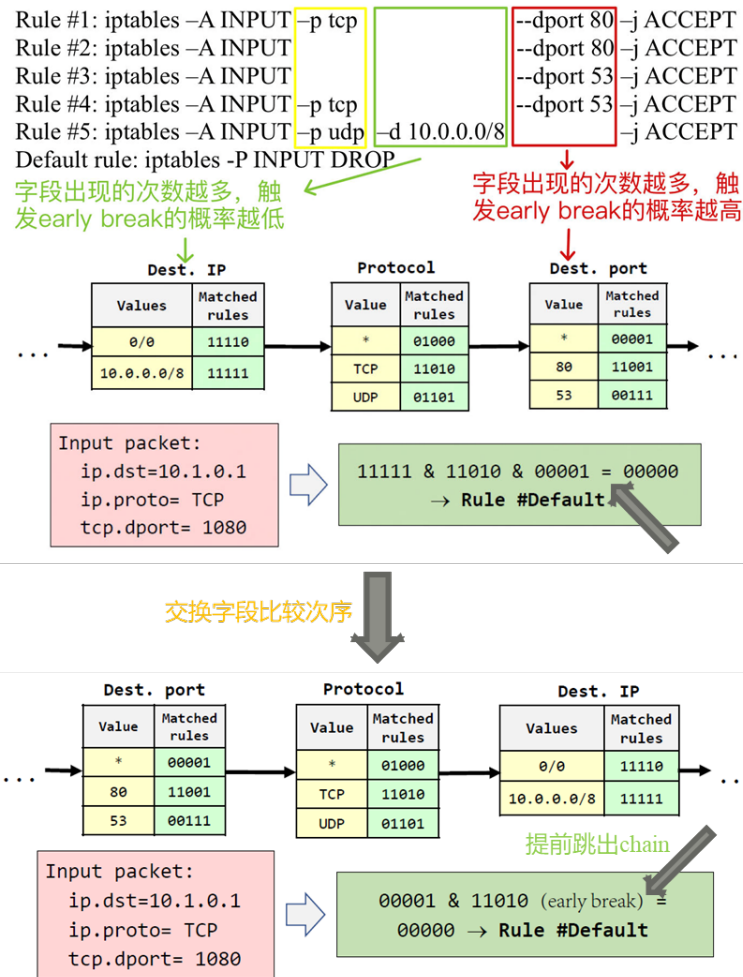


图 7: HORUS 改进前后示例

### 6.1.3 增加选项支持

除了对性能的改进以外，我对 bpf-iptables 的交互接口也作了一些改进，例如增加了支持替换某条过滤规则的选项 (-R/-replace)，并且修复了一些 Bug。值得注意的是，iptables 本身有四张表：RAW、FILTER、NAT 和 MANGLE，但是目前的 bpf-iptables 只支持 FILTER 表。

## 6.2 实验环境搭建

```
ubuntu@ubuntu1804: ~/Desktop/polycube-mod
File Edit View Search Terminal Help
ubuntu@ubuntu1804:~/Desktop/polycube-mod$ ./scripts/install.sh
Use 'install.sh -h' to show advanced installation options.
+ set -e
+++ dirname ./scripts/install.sh
++ cd ./scripts
++ pwd
+ DIR=/home/ubuntu/Desktop/polycube-mod/scripts
+ source scripts/pre-requirements.sh
++ '[' -z /home/ubuntu/Desktop/polycube-mod/scripts ']'
++ mkdir -p /home/ubuntu/dev
++ sudo apt update
```

图 8: 环境部署命令

实验环境基于 VMware 搭建的 Linux 发行版 Ubuntu 18.04.5 LTS，内核版本为 5.4。处理器为 Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz；图形处理器为 Intel(R) UHD Graphics 630；内存为 64GB DDR4。bpf-iptables 基于 polycube 提供的框架，因此要运行 bpf-iptables 必须安装整个 polycube。首先使用 git clone 拉取整个项目，然后进入项目目录并运行 ./scripts/install.sh 一键安装环境，如图 8 示。当显示如图 9 示的输出时表示 polycube 的环境已经搭建成功。

```
ubuntu@ubuntu1804: ~/Desktop/polycube-mod
File Edit View Search Terminal Help
-- Installing: /lib/systemd/system/polycubed.service
installing polycubectl ...

Installation completed successfully
-- Installing: /usr/share/bash-completion/completions/polycubectl
+ success_message
+ set +x

Installation completed successfully

You can now start the polycube daemon:
  manually:      sudo polycubed -d
  with systemd: sudo systemctl start polycubed
and then interact with it using the Polycube command line:
polycubectl -h
```

图 9: 环境部署成功

```
ubuntu@ubuntu1804: ~/Desktop/polycube-mod/build
File Edit View Search Terminal Help
ubuntu@ubuntu1804:~/Desktop/polycube-mod$ mkdir -p build/
ubuntu@ubuntu1804:~/Desktop/polycube-mod$ cd build/
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$ cmake .. -DENABLE_PCN_IPTABLES=ON
-- Version is HEAD-HASH-NOTFOUND+ [git: (branch/commit): NOT_FOUND/GITDIR-N]
Not a git repository
To compare two paths outside a working tree:
usage: git diff [--no-index] <path> <path>
fatal: not a git repository (or any of the parent directories): .git
CMake Warning at src/libs/bcc/CMakeLists.txt:33 (message):
  Failed to update submodule libbpf

-- Latest recognized Git tag is HEAD-HASH-NOTFOUND
-- Git HEAD is GITDIR-NOTFOUND
-- Revision is EAD-HASH-NOTFOUND-GITDIR-N
-- Found LLVM: /usr/lib/llvm-9/include 9.0.0 (Use LLVM_ROOT environment variable for another version of LLVM)
-- Could NOT find LibDebuginfod (missing: LIBDEBUGINFOD_LIBRARIES LIBDEBUGINFOD_INCLUDE_DIRS)
-- Using static-libstdc++
-- Could NOT find LuaJIT (missing: LUAJIT_LIBRARIES LUAJIT_INCLUDE_DIR)
-- jsoncons v0.142.0
-- The following OPTIONAL packages have been found:

* BISON
* FLEX
* Threads

-- The following REQUIRED packages have been found:

* LibYANG
* LLVM
* LibElf

-- The following OPTIONAL packages have not been found:

* LibDebuginfod
* LuaJIT

-- systemd services install dir: /lib/systemd/system
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ubuntu/Desktop/polycube-mod/build
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$ make -j`nproc` && sudo make install
```

图 10: 编译 bpf-iptables

接着使用如图 10 所示的命令编译 polycube，并注意需要添加 -ENABLE\_PCN\_IPTABLES=ON 启用 bpf-iptables。图 11 所示是编译成功的终端输出。

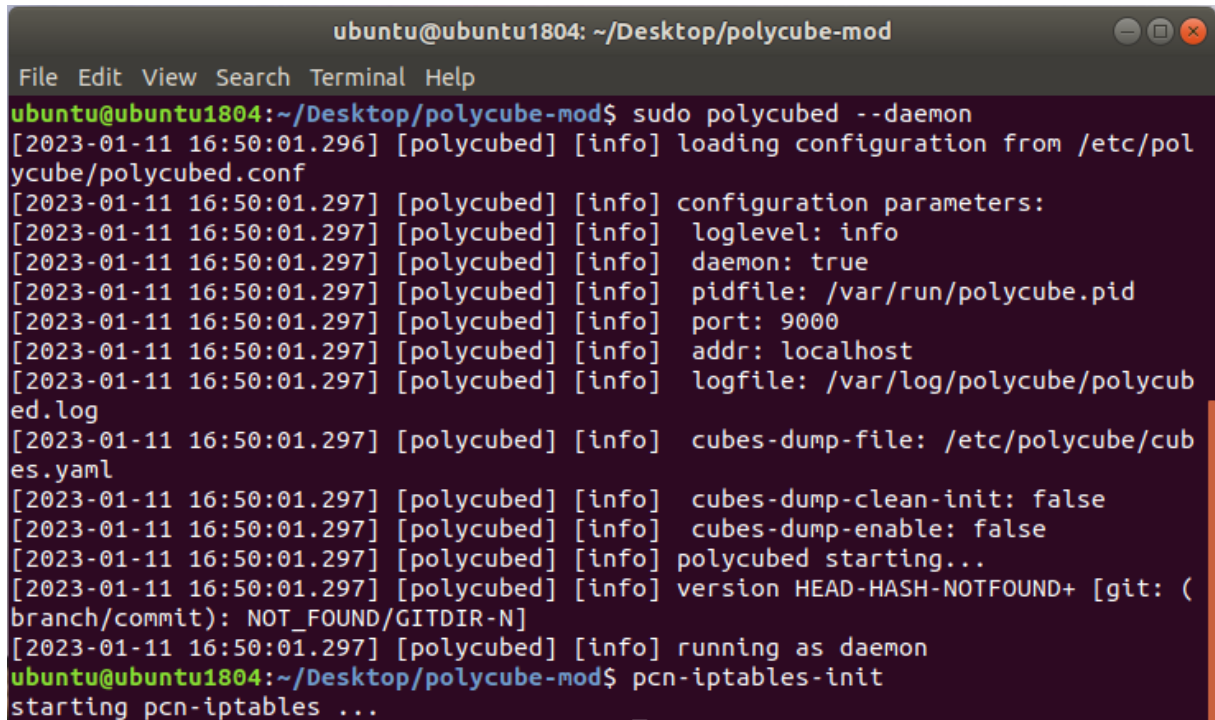
```
ubuntu@ubuntu1804: ~/Desktop/polycube-mod/build
File Edit View Search Terminal Help
-- Installing: /usr/lib/libpcn-iptables.so
-- Installing: /usr/lib/libpcn-transparenthelloworld.so
-- Installing: /usr/lib/libpcn-synflood.so
-- Installing: /usr/lib/libpcn-packetcapture.so
-- Installing: /usr/lib/libpcn-dynmon.so
-- Installing: /usr/lib/libpcn-k8sdispatcher.so
-- Installing: /usr/local/bin/polycubed
-- Set runtime path of "/usr/local/bin/polycubed" to ""
-- Installing: /lib/systemd/system/polycubed.service
installing polycubectl ...

Installation completed successfully
-- Installing: /usr/share/bash-completion/completions/polycubectl
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$
```

图 11: 编译 bpf-iptables

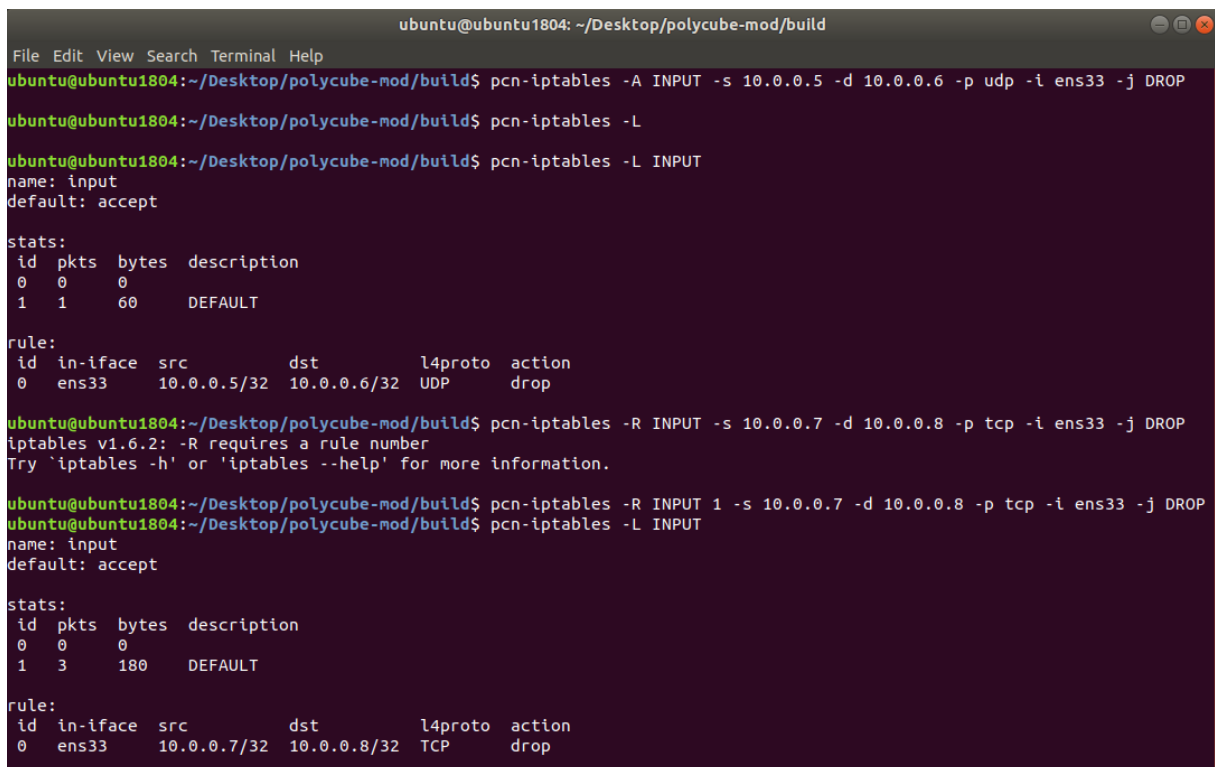
### 6.3 界面分析与使用说明

bpf-iptables 的使用和 iptables 的使用方式十分相似，本质上就是配置 iptables 的命令。但是由于 bpf-iptables 依赖于 polycube，因此需要先启动 polycube 的守护进程，启动的方式如图 12 所示。bpf-iptables 的使用示例如图 13 所示，包括追加、查看以及替换规则等操作。



```
ubuntu@ubuntu1804: ~/Desktop/polycube-mod
File Edit View Search Terminal Help
ubuntu@ubuntu1804:~/Desktop/polycube-mod$ sudo polycubed --daemon
[2023-01-11 16:50:01.296] [polycubed] [info] loading configuration from /etc/polycube/polycubed.conf
[2023-01-11 16:50:01.297] [polycubed] [info] configuration parameters:
[2023-01-11 16:50:01.297] [polycubed] [info] loglevel: info
[2023-01-11 16:50:01.297] [polycubed] [info] daemon: true
[2023-01-11 16:50:01.297] [polycubed] [info] pidfile: /var/run/polycube.pid
[2023-01-11 16:50:01.297] [polycubed] [info] port: 9000
[2023-01-11 16:50:01.297] [polycubed] [info] addr: localhost
[2023-01-11 16:50:01.297] [polycubed] [info] logfile: /var/log/polycube/polycubed.log
[2023-01-11 16:50:01.297] [polycubed] [info] cubes-dump-file: /etc/polycube/cubes.yaml
[2023-01-11 16:50:01.297] [polycubed] [info] cubes-dump-clean-init: false
[2023-01-11 16:50:01.297] [polycubed] [info] cubes-dump-enable: false
[2023-01-11 16:50:01.297] [polycubed] [info] polycubed starting...
[2023-01-11 16:50:01.297] [polycubed] [info] version HEAD-HASH-NOTFOUND+ [git: (branch/commit): NOT_FOUND/GITDIR-N]
[2023-01-11 16:50:01.297] [polycubed] [info] running as daemon
ubuntu@ubuntu1804:~/Desktop/polycube-mod$ pcn-iptables-init
starting pcn-iptables ...
```

图 12: 启动 polycubed 以及 bpf-iptables



```
ubuntu@ubuntu1804: ~/Desktop/polycube-mod/build
File Edit View Search Terminal Help
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$ pcn-iptables -A INPUT -s 10.0.0.5 -d 10.0.0.6 -p udp -i ens33 -j DROP
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$ pcn-iptables -L
name: input
default: accept
stats:
  id  pkts  bytes  description
  0   0     0
  1   1    60   DEFAULT
rule:
  id  in-iface  src          dst          l4proto  action
  0   ens33     10.0.0.5/32  10.0.0.6/32  UDP      drop
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$ pcn-iptables -R INPUT -s 10.0.0.7 -d 10.0.0.8 -p tcp -i ens33 -j DROP
iptables v1.6.2: -R requires a rule number
Try 'iptables -h' or 'iptables --help' for more information.
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$ pcn-iptables -R INPUT 1 -s 10.0.0.7 -d 10.0.0.8 -p tcp -i ens33 -j DROP
ubuntu@ubuntu1804:~/Desktop/polycube-mod/build$ pcn-iptables -L INPUT
name: input
default: accept
stats:
  id  pkts  bytes  description
  0   0     0
  1   3    180   DEFAULT
rule:
  id  in-iface  src          dst          l4proto  action
  0   ens33     10.0.0.7/32  10.0.0.8/32  TCP      drop
```

图 13: 使用 bpf-iptables 的示例

### 6.4 创新点

在前面已经提到，对 bpf-iptables 的改进包括 HORUS 集选取规则的改进，LBVS 算法中增加的自适应的字段比较次序，以及交互界面的改进。这些都是 bpf-iptables-mod 的创新点。

## 7 实验结果分析

实验从 System Benchmark 和 Micro Benchmark 两个方面对比了改进前的 bpf-iptables, 改进后的 bpf-iptables(bpf-iptables-mod) 以及 Ubuntu 自带的 iptables 的差异。这两个方面分别代表防火墙在宏观和微观两个角度的性能表现。

### 7.1 System Benchmark

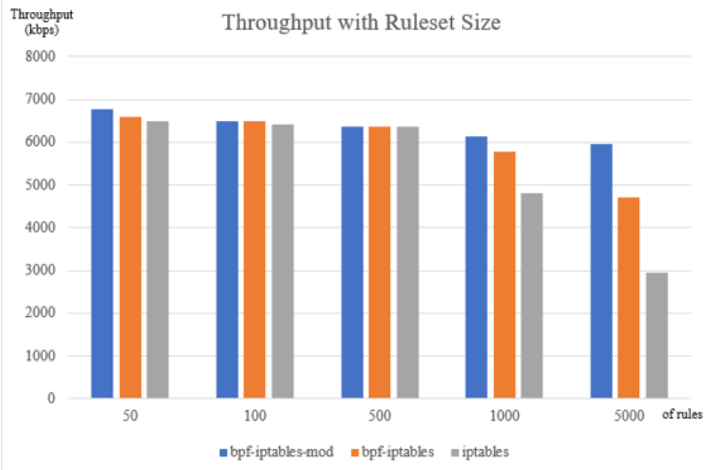


图 14: Throughput with Ruleset Size

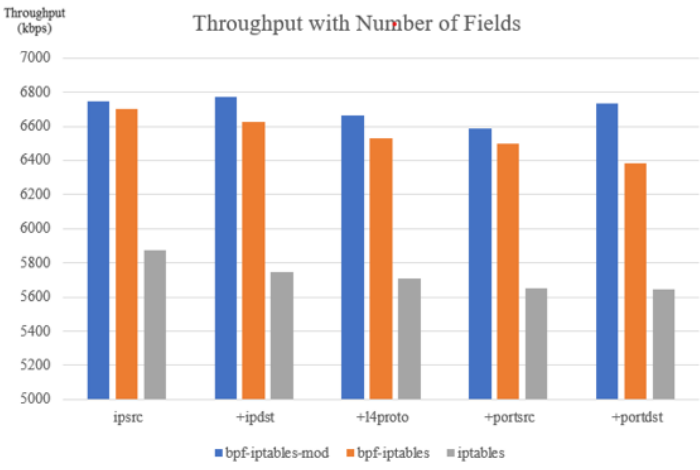


图 15: Throughput with Number of Fields

控制每条过滤规则有源 IP、目的 IP 和协议三个字段，测量三种 iptables 下网络吞吐量与过滤规则数量的关系。测试的 payload 为 HTTP 请求，每个请求的数据包大小固定为 854 字节，测试的结果如图 14所示。可以看到 bpf-iptables 比改进前有性能提升，但当过滤规则的数量比较少时提升并不明显，随着过滤规则数量的增加性能提升越明显。

另一方面，控制过滤规则的总数目固定为 1000，测试三种 iptables 下网络吞吐量与过滤规则字段数量的关系。测试的 payload 为 HTTP 请求，每个请求的数据包大小固定为 854 字节，逐渐增加的字段分别为源 IP、源端口、传输层协议、目的 IP 和目的端口，测试的结果如图 15所示。可以看到 bpf-iptables 比改进前有性能提升，但当过滤规则的字段数量比较少时提升并不明显，随着过滤规则字段数量的增加性能提升越明显。

### 7.2 Micro Benchmark

rules	ipt	bpf-iptables-mod			HORUS	
		t1	t2	t3	t <sub>H</sub> 1	t <sub>H</sub> 2
0	7	0.1600	1440	0.0811	0.0512	377
50	7	1.2629	1501	0.1764	0.7758	382
100	7	2.4394	1588	0.0770	0.9645	358
500	8	11.7675	1514	0.0748	4.1869	382
1000	9	24.7133	1749	0.0995	8.5766	444
5000	39	123.1120	1733	0.0867	17.4342	367

图 16: Running Time of Different Types

测试改进后的 bpf-iptables 的五种微观时间，分别为计算位向量、创建和加载新的链、删除旧的链、分析 HORUS 集及创建 HORUS 集，测试的结果如图 16所示。与此同时作为对比，也测量了 Ubuntu 自带的 iptables 的处理时间，测试的结果如图 16所示。



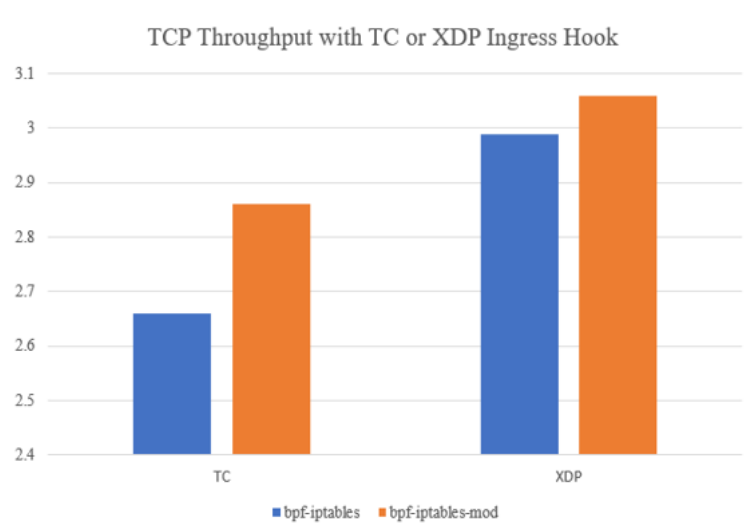


图 17: TCP Throughput with TC or XDP Ingress Hook

bpf-iptables 同时支持在 XDP 和 TC 网络接口卡两种模式下工作。XDP 在实现机制上比 TC 要更加优秀，但是并不是所有的网卡都支持 XDP 模式；即便网卡支持 XDP 模式，也只支持进入主机的数据包。测试改进前后的 bpf-iptables 在两种模式下的性能表现，测试结果如图 17 所示。图 17 中横坐标表示工作模式和程序类型，纵坐标表示实际最高的 TCP 吞吐量。实验环境中的链路带宽上限为 10Gbps。有实验结果可以看出，XDP 比 TC 有成倍性能的提升。

## 8 总结与展望

添加新的 eBPF helpers 绝对是一个合适的方向，特别是对于 bpf-iptables 的 eBPF conntrack，它远未完成并且仅支持基本场景。一个专门的 helper 可以实现更完整的功能，而不必处理 eBPF 程序众所周知的限制（例如，指令数量、循环）。bpf-iptables 面临的 eBPF 子系统的最大限制之一是允许的最大指令数，目前限制为 4K，从而导致支持的规则的最大数量限制为 8K。这方面，扩展 eBPF 验证器以支持有界循环将非常有帮助。

本文介绍了 bpf-iptables，一种基于 eBPF 的 Linux 防火墙，旨在保留 iptables 过滤语义同时提高其速度和可扩展性，特别是在使用大量规则时。bpf-iptables 能够在运行时向内核中动态编译和注入 eBPF 程序，以构建基于优化的数据路径。在实际的防火墙配置上，bpf-iptables 与 Linux 内核紧密集成可能比其他解决方案（例如 DPDK）具有更大的优势，因为可以与其它的内核功能（例如路由）和 Linux 生态系统的其它工具合作。此外，bpf-iptables 不需要自定义内核模块或其他软件框架，在某些情况下（例如公共数据中心）是不允许的。与现有解决方案相比，bpf-iptables 保证了巨大的性能优势，特别是在大量的情况下过滤规则；此外，当没有规则被实例化时，bpf-iptables 不会在系统引入过度的开销，即使在某些情况下在入口挂钩上使用 XDP 可能会损害系统的整体性能。现有的 eBPF 限制可以通过临时工程选择（例如，分类管道）和巧妙的优化（例如，HORUS）规避，保证了进一步的可扩展性。

我的工作是在完全理解整个 bpf-iptables 的框架的条件下，对 bpf-iptables 的改进包括 HORUS 集选取规则的改进，LBVS 算法中增加的自适应的字段比较次序，以及交互界面的改进，对于过滤规则的数量比较大时有一定的性能提升。另外，我认为 bpf-iptables 远远未达到 iptables 的功能。除了需要对于 RAW、MANGLE 以及 NAT 表的支持以及实现转发的相关功能。



## 参考文献

- [1] GRAF T. Why is the kernel community replacing iptables with BPF?[EB/OL]. 2018 [2022-12-24]. <https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables/>.
- [2] VIEIRA M A, CASTANHO M S, PACÍFICO R D, et al. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications[J]. ACM Computing Surveys (CSUR), 2020, 53(1): 1-36.
- [3] HØILAND-JØRGENSEN T, BROUER J D, BORKMANN D, et al. The express data path: Fast programmable packet processing in the operating system kernel[C]//Proceedings of the 14th international conference on emerging networking experiments and technologies. 2018: 54-66.
- [4] BORKMANN D. net: add bpfILTER[EB/OL]. 2018 [2022-12-24]. <https://lwn.net/Articles/747504/>.
- [5] Netronome. BPF, eBPF, XDP and Bpfilter...What are These Things and What do They Mean for the Enterprise?[EB/OL]. 2018 [2022-12-24]. <https://www.netronome.com/blog/bpf-ebpf-xdp-and-bpfilter-what-are-these-things-and-what-do-they-mean-enterprise/>.
- [6] MIANO S, BERTRONE M, RISSO F, et al. Creating complex network services with ebpf: Experience and lessons learned[C]//2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR). 2018: 1-8.
- [7] SRINIVASAN V, VARGHESE G, SURI S, et al. Fast and scalable layer four switching[C]//Proceedings of the ACM SIGCOMM'98 conference on Applications, technologies, architectures, and protocols for computer communication. 1998: 191-202.
- [8] DALY J, TORNG E. Tuplemerge: Building online packet classifiers by omitting bits[C]//2017 26th International Conference on Computer Communication and Networks (ICCCN). 2017: 1-10.
- [9] LAKSHMAN T, STILIADIS D. High-speed policy-based packet forwarding using efficient multi-dimensional range matching[J]. ACM SIGCOMM Computer Communication Review, 1998, 28(4): 203-214.