



SOLANA

# 使用 Rust 编写 BPF 链上程序

讲者: Justin Starry

# 自我介绍



Github: [@jstarry](#)

- 两年前开始学习 Rust
- 我专门做客户端应用开发
- 我做的第一个 Rust 项目是 Yew 框架的网路应用
- 做完后, 我负责维护 Yew 框架



- 创建于 2017 年下半年
- 30名员工
- 总部在旧金山
- 高速单层区块链网络
  - 支持智能合约(叫做链上程序)
  - 目前最大 TPS(每秒执行交易量)可以达到 50000
    - 以太坊(Ethereum)只有 15 左右
  - 用 Rust 编写的
  - 提供 BPF 虚拟机来执行链上程序



1. BPF(伯克利包过滤器)是什么？
2. Solana 为什么选择 BPF？
3. Solana 怎么定制了 BPF
4. Rust 工具链的挑战
5. 介绍 Rust 的 BPF 链上程序



# Solana 虚拟机字节码

## 需求

- 校验安全和执行字节码, 要求速度既快又 稳定
- 有效地即时编译到 x86 架构
- 可移植性

## 为什么没有用 WebAssembly(Wasm)?

- 速度快, 但是 Wasm 优化了程序字节码大小
- 而 BPF 更注重字节码运行的性能
- Wasm 的基于栈虚拟机的即时编译器更复杂



# BPF 简介

- BPF 技术包括指令集、虚拟机、安全校验机制
- BPF 一开始用来过滤 Linux 内核中的数据包
- 现在更灵活, 用户可以加载他们写的 BPF 程序到内核中
- 速度和安全是 Linux 的基本需求
- 为了高性能, 所有的现代架构都提供 BPF 程序的即时编译器(JIT compiler)
- BPF 目前的几种用途包括:
  - 安全和网络
  - 观测和监控
  - 追踪和性能分析(可以分析 Linux 内核和用户程序)



# BPF 架构

- 有限指令集
  - 大部分指令能够直接映射到 x86
- 11个64位寄存器(也可以作为32位子寄存器)
  - R0 存放被调用的函数的返回值
  - R1-R5 存放函数参数
  - R6-R9 是被调用方保存(callee saved)的寄存器
  - R10 存放只读栈帧指针(frame pointer) 地址
- 一个程序计数器



# BPF 的局限性

- Linux 的 BPF 校验器必须证明用户 bpf 程序没有不良的行为
  - 需要停止 (禁止后向跳转)
  - 只允许有界循环
  - 不可以跳转到边界外内存地址
- 最大指令数限制在 1 百万条以内
- 每个函数能用最多 5 个参数
- 目前上限是 32 层调用
- 固定 512 字节大小的栈帧
- 没有带符号数的除法 (signed division):(  
(还没)提供共享库链接的功能





# Solana 定制化 BPF

- 由于 Linux BPF 校验器采用 GPL 许可证, 我们没法用
  - 没有停止检测, 运行时中限制指令数量
  - 内存访问在运行时检查
- 增加了栈帧大小, 从 512 字节增加到 4096 字节
- 启用了链接共享库
- 去掉了5个参数的限制(第 6 个以上的参数通过栈帧传递)
- 最多指令数量只可以达到 20万
- 加倍了最大调用深度(从 32 加到 64)



# Solana BPF Rust 工具链

- LLVM 已经提供 Rust 前端和 BPF 后端
  - 分叉了 LLVM 项目来改变 BPF 后端
    - 增加了栈帧大小
    - 每个函数可以接受超 5 个参数
    - 函数可以访问其他函数的栈帧
- Rust 编译器不官方支持 BPF 目标
  - 为了添加 BPF 目标分叉了 rustc (Rust 编译器)
  - Cargo 还没支持编译一个用户提供的 Rust 标准库
  - 从标准库中删除了以下功能:网络、线程、I/O、随机、等。
  - 简单化 Panic 行为 (没有提供栈追踪)
  - 用了 Xargo 来编译被 Solana 定制的 Rust 标准库
- 刚才开发了测试版的 x86 即时编译器(JIT compiler)
- 提供 cargo 子命令: `cargo build-bpf`



# Solana 链上程序

- 目前支持 Rust 和 C 语言
- 单入口
- 提供叫做 “syscalls”(系统调用):
  1. 调用其他链上程序
  2. 调用被本地编译的密码函数 (sha256)
  3. 调用记录器
- 链上程序被 “compute units”(算单位)的数量限制
  - 每 BPF 指令要花一个算单位
  - 每系统调用有自己的价格



# Solana 链上程序的例子

```
7 solana_program::entrypoint!(process_instruction);
8 fn process_instruction(
9     program_id: &Pubkey, // 链上程序的公钥
10    accounts: &[AccountInfo], // 账号列表
11    instruction_data: &[u8],
12 ) -> ProgramResult {
13
14     // 你要打个招呼的账号
15     let account: &AccountInfo = &accounts[0];
16
17     // 你要打几个招呼
18     let num_times: u32 = instruction_data[0] as u32;
19     for _ in 0..num_times {
20         info!("你好!");
21     }
22
23     // 添加并存储该账号被打过几个招呼
24     let mut data: RefMut<&mut [u8]> = account.try_borrow_mut_data()?;
25     let past_greetings: u32 = LittleEndian::read_u32(buf: &data);
26     LittleEndian::write_u32(buf: &mut data[0..], n: past_greetings + num_times);
27
28     info!("再见!");
29
30     Ok(())
31 }
```

### 参考列表

- BPF 文档: <https://www.kernel.org/doc/html/latest/bpf>
- BPF 设计常问问题: [https://www.kernel.org/doc/html/latest/bpf/bpf\\_design\\_QA.html](https://www.kernel.org/doc/html/latest/bpf/bpf_design_QA.html)
- Cilium BPF 参考指南: <https://docs.cilium.io/en/v1.9/bpf/>
- eBPF 的全面介绍: <https://lwn.net/Articles/740157/>
- eBPF: <https://ebpf.io/>
- 非官方 eBPF 规格: <https://github.com/iovisor/bpf-docs/blob/master/eBPF.md>
- BPF: <https://www.kernel.org/doc/Documentation/networking/filter.txt>

### 社区 Rust 项目

- eBPF的虚拟机和即时编译: <https://github.com/qmonnet/rbpf>
- Sysroot 交叉编译器: <https://github.com/japaric/xargo>

Solana BPF 虚拟机：

- <https://github.com/solana-labs/rbpf>

Solana Rust 工具链：

- 定义 rustc 分叉: <https://github.com/solana-labs/rust>
- 改性 Rust sysroot: <https://github.com/solana-labs/rust-bpf-sysroot>
- LLVM 分叉(有定义 BPF 后端) <https://github.com/solana-labs/llvm-project>

试试看！

- <https://github.com/solana-labs/example-helloworld>


跟我们聊天！

- <https://solana.com/discord>



加我 WeChat



Justin @ Solana 

加拿大 多倫多

