Outreachy – Linux Networking

Netdev 0x16, 2022

Roopa Prabhu (Nvidia), Stefano Brivio (Red Hat), Jaehee Park

Agenda

- Goals
- Outreachy intro – Roopa Prabhu
- Networking projects @ Outreachy – Roopa Prabhu
- Further project ideas and half-done things – Stefano Brivio
- Hear from our Intern – Jaehee Park
- Resources

Goal for this talk

- Outreach
- Increase networking participation in Outreachy
  - Kernel, userspace, anything networking ...
- Call for more mentors, interns
- Call for more projects
- Call for more sponsors

Outreachy Intro

- Outreachy's goal is to increase diversity in Open Source
- Outreachy https://www.outreachy.org/ provides paid remote internships in open source and open science
  - *to people subject to systemic bias and impacted by underrepresentation in the tech industry*
- Two rounds per year
  - *December and May*
  - *3 months full time internships*
- Many open source communities to pick from - Linux kernel is one of them!
  - Linux kernel outreachy coordinator is Alison Schofield
  - https://www.outreachy.org/apply/project-selection/
- Sponsors sponsor outreachy communities
- Mentors can submit projects to outreachy communities

Outreachy Kernel Networking so far

2020

- Interns: *Briana Oursler, Lourdes Pedrajas*
- Mentors: *Stefano Brivio, Davide Caratti*
- Projects:
  - deduplication in networking selftests script: turned into half-baked library draft
  - new *tc* selftests and improving the existing ones

2022

- Interns: *Jaehee Park, Sevinj Aghayeva, Alaa Mohamed*
- Mentors: *Roopa Prabhu, Stefano Brivio, Andy Roulin*
- Projects:
  - Linux kernel neighbour subsystem fix
  - Linux bridge driver fixes
  - netlink extack fixes
  - Linux kernel selftests and fixes
  - Linux kernel selftest library

Half-done things

- Three months are not that long
- Not completed yet (Jaehee will show some bits)
  - Library for selftests
    - ...it's complicated.
  - Tool to embed kselftests in initramfs, ~10 seconds build-to-run in VM
    - more than half-done (Sevinj did a lot), usable for many areas
    - https://mbuto.sh
  - AVX2-based clearing of pages (first XDP, then slab, then the world!)
    - Based on some old ideas by Stefano and Jesper Brouer
    - Jaehee started checking out things just before end of internship

Next Outreachy round?

- Plan to finish the kselftests library in the next round (“May” 2023)
- And perhaps something else. New ideas and mentors welcome!
  - ...don’t let Stefano just POSIXify interns.
- Deadlines (approximate):
  - submission of new projects in late March 2023
  - mentor sign-up around that date
  - contribution phase starts after that
    - ...yes, that’s when you see people fixing whitespace in decnet and pcmcia
    - we’re trying to improve that

Hear from Outreachy intern - Jaehee Park!

Brief intro

- Software engineer interested in embedded applications, the Linux kernel, networking, debugging, testing, and more!

- Outreachy ‘22 intern

- Documented Outreachy experience in blog
  - https://jhpark1013.github.io/blog/

Outreachy Linux kernel internship

Mentored by Roopa Prabhu,
Stefano Brivio, and Andy Roulin

1. Projects
   - Project 1: New subnet filtering feature added to ARP and ndisc in the kernel networking stack
   - Project 2: exploring XDP with task example (zero network buffer pages using AVX2 instructions and re-evaluate performance)
2. Tooling
   - perf: debugging tool
   - mbuto: builds minimal image for kernel selftests
     - Sevinj Aghayeva Outreachy ‘22 intern contributed to kselftest feature
3. Networking tests
   - selftests
   - Linux kernel testing library for networking
     (Trying to factor common tasks from kernel networking selftests into a library)
     - Alaa Soliman Mohamed Outreachy ‘22 intern contributed as well as previous outreachy interns Stefano mentored

Project 1: Linux kernel patches
New feature in neighbor discovery

- Patches were sent to the **net-next** tree

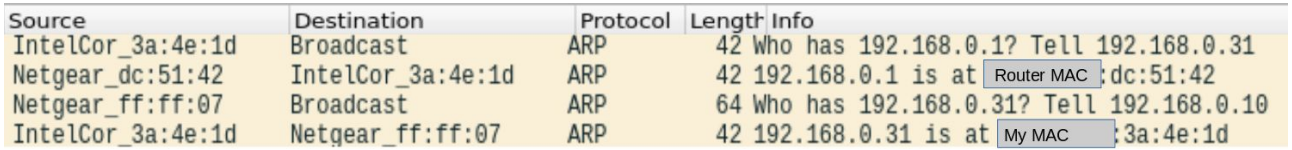
- Goal was to provide a subnet filtering option for **garp** during neighbor discovery

IPv4 vs IPv6 analogies

version	protocol	response	sysctl
IPv4	ARP (address resolution protocol)	garp (gratuitous ARP)	arp_accept
IPv6	ndisc / NDP (neighbor discovery)	unsolicited NA (neighbor advertisement)	drop_unsolicited_na & accept_untracked_na

Basic networking concepts

Arp communication



Garp (gratuitous arp)

- A device announces itself without being prompted by an ARP request
- It’s also called unsolicited advertisements in IPv6
- By default, devices are configured to drop all gratuitous ARP (garp) frames. But you can enable the device to accept garp with the arp_accept sysctl

Subnet filtering in ARP: new knob in arp_accept

- The subnet filtering option has been added as the 3rd “knob” to the arp_accept sysctl
- Instead of calling IN_DEV_ARP_ACCEPT directly, define the arp_accept() function to include more than just 2 features
- The arp_accept() function includes switch statement to output 0 or 1 based on the conditions

```
static int arp_accept(struct in_device *in_dev, __be32 sip)
{
    struct net *net = dev_net(in_dev->dev);
    int scope = RT_SCOPE_LINK;

    switch (IN_DEV_ARP_ACCEPT(in_dev)) {
    case 0: /* Don't create new entries from garp */
        return 0;
    case 1: /* Create new entries from garp */
        return 1;
    case 2: /* Create a neighbor in the arp table only if sip
         * is in the same subnet as an address configured
         * on the interface that received the garp message
         */
        return !!inet_confirm_addr(net, in_dev, sip, 0, scope);
    default:
        return 0;
    }
}
```

Subnet filtering in ndisc: new knob in accept_untracked_na

- RFC 9131 adds a behavior to accept unsolicited NA
- Subnet filtering added as the 3rd knob to this new accept_untracked_na sysctl

pseudocode:

```
static void ndisc_recv_na(struct sk_buff *skb)
{
        if (/* condition */) {
                /* don't accept */
                return;
        }

        ...
        /* bunch of if...return conditions  */
        ...

        /* Include code to accept untracked_na here */

        /* If we make it to here, and neighbor exists,
         * update cache
         */
        neigh = neigh_lookup(&nd_tbl, &msg->target, dev);
        if(neigh) {
                ndisc_update(...);
        }
}
```

```
neigh = neigh_lookup(&nd_tbl, &msg->target, dev);

/* RFC 9131 updates original Neighbour Discovery RFC 4861.
 * NAs with Target LL Address option without a corresponding
 * entry in the neighbour cache can now create a STALE neighbour
 * cache entry on routers.
 *
 *   entry accept  fwding  solicited        behaviour
 * ------- ------  ------  ---------    ----------------------
 * present      X       X          0     Set state to STALE
 * present      X       X          1     Set state to REACHABLE
 *  absent      0       X          X     Do nothing
 *  absent      1       0          X     Do nothing
 *  absent      1       1          X     Add a new STALE entry
 *
 * Note that we don't do a (daddr == all-routers-mcast) check.
 */
new_state = msg->icmph.icmp6_solicited ? NUD_REACHABLE : NUD_STALE;
if (!neigh && lladdr && idev && idev->cnf.forwarding) {
        if (accept_untracked_na(dev, saddr)) {
                neigh = neigh_create(&nd_tbl, &msg->target, dev);
                new_state = NUD_STALE;
        }
}
```

Subnet filtering in ndisc: new knob in accept_untracked_na

```
static int accept_untracked_na(struct net_device *dev, struct in6_addr *saddr)
{
    struct inet6_dev *idev = __in6_dev_get(dev);

    switch (idev->cnf.accept_untracked_na) {
    case 0: /* Don't accept untracked na (absent in neighbor cache) */
       return 0;
    case 1: /* Create new entries from na if currently untracked */
       return 1;
    case 2: /* Create new entries from untracked na only if saddr is in the
        * same subnet as an address configured on the interface that
        * received the na
        */
       return !!ipv6_chk_prefix(saddr, dev);
    default:
       return 0;
    }
}
```

Project 2 - Exploring XDP with a task example: zero network buffer pages using AVX2 instructions

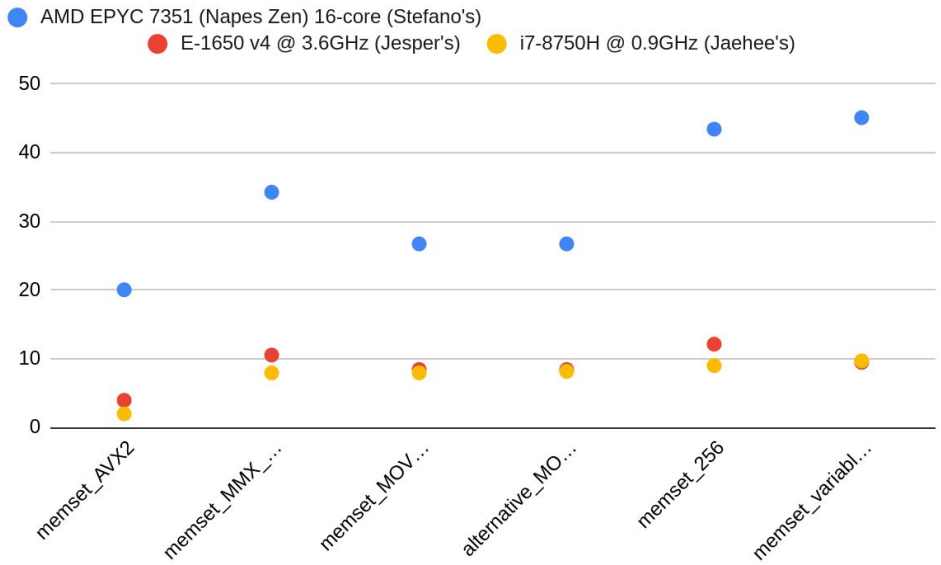
- Benchmarking code execution time inside the kernel
- Testing clearing pages with various SIMD operations (like MMX and AVX2)
- Using prototype-kernel by Jesper Brouer as testbed for experiments
  - netoptimizer / prototype-kernel
- Goal was to experiment with tests in prototype-kernel

```
static int time_memset_avx2_256(struct time_bench_record *rec, void *data)
{
#define CONST_CLEAR_SIZE 256
    int i, j;
    uint64_t loops_cnt = 0;

    time_bench_start(rec);

    for (i = 0; i < rec->loops; i++) {
        kernel_fpu_begin();
        TIME_MEMSET_AVX2_ZERO(0);

        loops_cnt++;
        barrier();
        for (j = 0; j < BYTES_TO_YMM(CONST_CLEAR_SIZE); j++)
               TIME_MEMSET_AVX2_STORE(global_buf[YMM_BYTES * j], 0);
        barrier();

        kernel_fpu_end();
    }

    time_bench_stop(rec, loops_cnt);
    return loops_cnt;
#undef  CONST_CLEAR_SIZE
}
```

Project 2 - Exploring XDP with a task example: zero network buffer pages using AVX2 instructions

Measurements in nanoseconds

	AMD EPYC 7351 (Napes Zen) 16-core (Stefano's)	E-1650 v4 @ 3.6GHz (Jesper's)	i7-8750H @ 0.9GHz (Jaehee's)
memset_AVX2	20.049	3.954	1.983
memset_MMX_256	34.251	10.544	7.945
memset_MOVQ_256	26.731	8.435	7.956
alternative_MOVQ_256	26.73	8.433	8.164
memset_256	43.437	12.126	9.001
memset_variable_step	45.107	9.489	9.689



AVX2 can clear pages on XDP paths faster!

Outreachy Linux kernel internship

Mentored by Roopa Prabhu,
Stefano Brivio, and Andy Roulin

1. Projects
   - New **subnet filtering feature** added to arp and ndisc in the kernel networking stack
   - Exploring XDP with task example
2. Tooling
   - perf: debugging tool
   - mbuto: builds minimal image for kernel selftests
3. Networking tests
   - selftests
   - Linux kernel testing **library** for networking

Debugging with perf

- Setup perf to work with the newest kernel to test my new patch
- I found an error in my selftest with perf!
- My neighbor cache table wasn't being updated (but on perf I saw neigh_update being called). So I knew there was a bug in my selftest

see the neigh_update being called
(and other calls in the network stack):

```
arping 46883 [003] 22325.771054: probe:neigh_update: (ffffffff
        ffffffffa47d3d11 neigh_update+0x1 ([kernel.kallsyms])
        ffffffffa48a7b70 arp_rcv+0x190 ([kernel.kallsyms])
        ffffffffa47c7fdf __netif_receive_skb_one_core+0x8f ([k
        ffffffffa47c8038 __netif_receive_skb+0x18 ([kernel.kal
        ffffffffa47c8279 process_backlog+0xa9 ([kernel.kallsym
        ffffffffa47c98e1 __napi_poll+0x31 ([kernel.kallsyms])
        ffffffffa47c9dbf net_rx_action+0x23f ([kernel.kallsyms
        ffffffffa4e000cf __softirqentry_text_start+0xcf ([kern
        ffffffffa3eaa666 do_softirq+0x66 ([kernel.kallsyms])
        ffffffffa3eaa6d0 __local_bh_enable_ip+0x50 ([kernel.ka
        ffffffffa47c6738 __dev_queue_xmit+0x388 ([kernel.kalls
        ffffffffa47c6da0 dev_queue_xmit+0x10 ([kernel.kallsyms
        ffffffffa4958fdc packet_snd+0x34c ([kernel.kallsyms])
        ffffffffa495ab16 packet_sendmsg+0x26 ([kernel.kallsyms
        ffffffffa479d6a5 sock_sendmsg+0x65 ([kernel.kallsyms])
        ffffffffa479e933 __sys_sendto+0x113 ([kernel.kallsyms]
        ffffffffa479e9d9 __x64_sys_sendto+0x29 ([kernel.kallsy
        ffffffffa49f5f01 do_syscall_64+0x61 ([kernel.kallsyms]
        ffffffffa4c0007c entry_SYSCALL_64_after_hwframe+0x44 (
            7fe4b976568a __libc_sendto+0x1a (/usr/lib/x86_64-l
         1000608d4dd5341 [unknown] ([unknown])
```

Trying out **mbuto**: building quick initramfs for VM images

(now with kselftests inside!)

- Originally, quick hack Stefano wrote to create small initramfs images for qemu
- Kselftests support added by Sevinj Aghayeva (Outreachy ‘22 intern)
- Runs from kernel source directory, builds a minimalistic VM image, & sources selftests and required host tools
- Significantly **streamlines** the testing workflow!
- Check out the demo here! https://mbuto.sh

```
kvm -kernel arch/x86/boot/bzImage
-initrd $(../mbuto/mbuto -p kselftests -C timens)
```

```
Press s for shell, any other key to run selftests

TAP version 13
1..7
# selftests: /timens: timens
# 1..10
# ok 1 Passed for CLOCK_BOOTTIME (syscall)
# ok 2 Passed for CLOCK_BOOTTIME (vdso)
# ok 3 Passed for CLOCK_BOOTTIME_ALARM (syscall)
# ok 4 Passed for CLOCK_BOOTTIME_ALARM (vdso)
# ok 5 Passed for CLOCK_MONOTONIC (syscall)
# ok 6 Passed for CLOCK_MONOTONIC (vdso)
# ok 7 Passed for CLOCK_MONOTONIC_COARSE (syscall)
# ok 8 Passed for CLOCK_MONOTONIC_COARSE (vdso)
# ok 9 Passed for CLOCK_MONOTONIC_RAW (syscall)
# ok 10 Passed for CLOCK_MONOTONIC_RAW (vdso)
# # Totals: pass:10 fail:0 xfail:0 xpass:0 skip:0 error:0
ok 1 selftests: /timens: timens
# selftests: /timens: timerfd
# 1..3
# ok 1 clockid=7
# ok 2 clockid=1
# ok 3 clockid=9
# # Totals: pass:3 fail:0 xfail:0 xpass:0 skip:0 error:0
# 1..3
ok 2 selftests: /timens: timerfd
# selftests: /timens: timer
# 1..3
# ok 1 clockid=7
# ok 2 clockid=1
# ok 3 clockid=9
# # Totals: pass:3 fail:0 xfail:0 xpass:0 skip:0 error:0
```

...

```
# 1..2
ok 7 selftests: /timens: futex
All tests passed, shutting down guest...
[   77.942829] sysrq: Power Off
[   77.944800] reboot: Power down
```

Focus on testing just the relevant set of tests that I’m interested in using the -C flag

(here, we’re just testing timens)

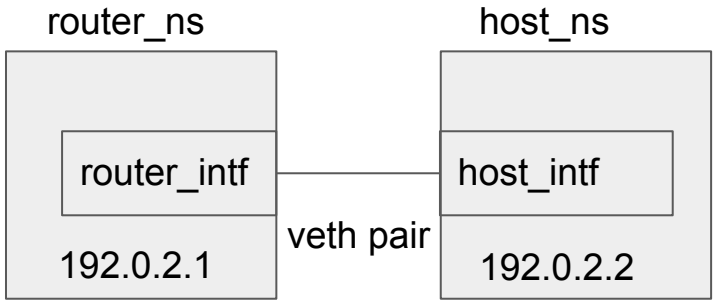
Outreachy Linux kernel internship

Mentored by Roopa Prabhu,
Stefano Brivio, and Andy Roulin

1. Projects
   - New **subnet filtering feature** added to arp and ndisc in the kernel networking stack
   - Exploring XDP with task example
2. Tooling
   - perf: debugging tool
   - mbuto: builds minimal image for kernel selftests
3. Networking tests
   - selftests
   - Linux kernel testing **library** for networking

Setup function (w/o using library)

```
setup() {
    local arp_accept=$1

    # Set up two namespaces
    ip netns add ${ROUTER_NS}
    ip netns add ${HOST_NS}

    # Set up interfaces veth0 and veth1, which are pairs in separate
    # namespaces. veth0 is veth-router, veth1 is veth-host.
    # first, set up the inteface's link to the namespace
    # then, set the interface "up"
    ip netns exec ${ROUTER_NS} ip link add name ${ROUTER_INTF} \
         type veth peer name ${HOST_INTF}

    ip netns exec ${ROUTER_NS} ip link set dev ${ROUTER_INTF} up
    ip netns exec ${ROUTER_NS} ip link set dev ${HOST_INTF} netns ${HOST_NS}

    ip netns exec ${HOST_NS} ip link set dev ${HOST_INTF} up
    ip netns exec ${ROUTER_NS} ip addr add ${ROUTER_ADDR}/${SUBNET_WIDTH} \
         dev ${ROUTER_INTF}

    ip netns exec ${HOST_NS} ip addr add ${HOST_ADDR}/${SUBNET_WIDTH} \
         dev ${HOST_INTF}
    ip netns exec ${HOST_NS} ip route add default via ${HOST_ADDR} \
         dev ${HOST_INTF}
    ip netns exec ${ROUTER_NS} ip route add default via ${ROUTER_ADDR} \
         dev ${ROUTER_INTF}

    ROUTER_CONF=net.ipv4.conf.${ROUTER_INTF}
    ip netns exec ${ROUTER_NS} sysctl -w \
         ${ROUTER_CONF}.arp_accept=${arp_accept} >/dev/null 2>&1
}
```

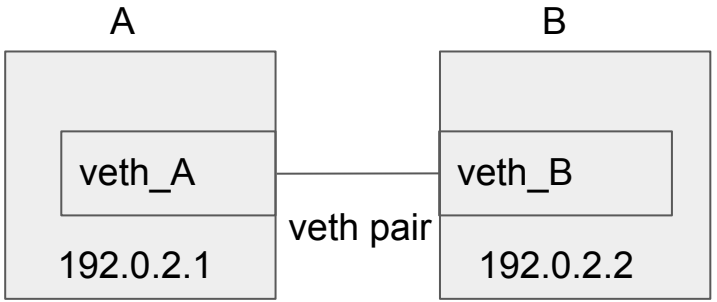


After setup, I use this “arping” command:

ip netns exec ${HOST_NS} arping -A -U ${**HOST_ADDR**}

Selftest using the library

```
#!/bin/sh

# Kselftest framework requirement - SKIP code is 4.
KSELFTEST_SKIP=4
. ./lib/util.sh

lib ns_vif routing link || return $KSELFTEST_SKIP
lib mtu tunnels       || return $KSELFTEST_SKIP


cleanup() {
        ip netns del A
        ip netns del B
}

setup_arp_test() {
    ns A B     || return $KSELFTEST_SKIP
    veth A B   || return $KSELFTEST_SKIP
}

arp_test_gratuitous() {
        local arp_accept=$1
        setup_arp_test
        ip netns exec B arping -A -U addr_get veth_B 4
}


cleanup
arp_test_gratuitous
```



Setup namespaces A and B

Interfaces veth_A and veth_B are pairs in namespaces A and B

Use **addr_get**() to get the IP address given the device interface.

Selftest w/o using library vs Selftest using library

```
ROUTER_NS="ns-router"
ROUTER_NS_V6="ns-router-v6"
ROUTER_INTF="veth-router"
ROUTER_ADDR="10.0.10.1"
ROUTER_ADDR_V6="2001:db8:abcd:0012::1"

HOST_NS="ns-host"
HOST_NS_V6="ns-host-v6"
HOST_INTF="veth-host"
HOST_ADDR="10.0.10.2"
HOST_ADDR_V6="2001:db8:abcd:0012::2"

SUBNET_WIDTH=24
PREFIX_WIDTH_V6=64

cleanup() {
    ip netns del ${HOST_NS}
    ip netns del ${ROUTER_NS}
}

cleanup_v6() {
    ip netns del ${HOST_NS_V6}
    ip netns del ${ROUTER_NS_V6}
}

setup() {
    local arp_accept=$1

    # Set up two namespaces
    ip netns add ${ROUTER_NS}
    ip netns add ${HOST_NS}

    # Set up interfaces veth0 and veth1, which are pairs in separate
    # namespaces. veth0 is veth-router, veth1 is veth-host.
    # first, set up the inteface's link to the namespace
    # then, set the interface "up"
    ip netns exec ${ROUTER_NS} ip link add name ${ROUTER_INTF} \
              type veth peer name ${HOST_INTF}

    ip netns exec ${ROUTER_NS} ip link set dev ${ROUTER_INTF} up
    ip netns exec ${ROUTER_NS} ip link set dev ${HOST_INTF} netns ${HOST_NS}

    ip netns exec ${HOST_NS} ip link set dev ${HOST_INTF} up
    ip netns exec ${ROUTER_NS} ip addr add ${ROUTER_ADDR}/${SUBNET_WIDTH} \
              dev ${ROUTER_INTF}

    ip netns exec ${HOST_NS} ip addr add ${HOST_ADDR}/${SUBNET_WIDTH} \
              dev ${HOST_INTF}
    ip netns exec ${HOST_NS} ip route add default via ${HOST_ADDR} \
              dev ${HOST_INTF}
    ip netns exec ${ROUTER_NS} ip route add default via ${ROUTER_ADDR} \
              dev ${ROUTER_INTF}

    ROUTER_CONF=net.ipv4.conf.${ROUTER_INTF}
    ip netns exec ${ROUTER_NS} sysctl -w \
              ${ROUTER_CONF}.arp_accept=${arp_accept} >/dev/null 2>&1
}
```

Much cleaner and shorter!

Using the library extracts the common code from tests

We eliminate repetitive code in net-next tests when setting up network topologies using the library

```
#!/bin/sh

# Kselftest framework requirement - SKIP code is 4.
KSELFTEST_SKIP=4
. ./lib/util.sh

lib ns_vif routing link || return $KSELFTEST_SKIP
lib mtu tunnels       || return $KSELFTEST_SKIP


cleanup() {
          ip netns del A
          ip netns del B
}

setup_arp_test() {
    ns A B || return $KSELFTEST_SKIP
    veth A B  || return $KSELFTEST_SKIP
}

arp_test_gratuitous() {
          local arp_accept=$1
          setup_arp_test
          ip netns exec B arping -A -U addr_get veth_B 4
}


cleanup
arp_test_gratuitous
```

Summary

- Learned a ton!
- Mentors helped discuss various ideas and improve on 2-3 iterations of the patchsets
- Working with mentors gave me confidence in interacting with the Linux kernel community
- Connections made in the open-source community and with the mentors is incredibly valuable!

Resources

- https://www.outreachy.org/
- https://www.outreachy.org/apply/project-selection/
- https://www.outreachy.org/sponsor/