**运行步骤**：

**配置：**

代码支持通过命令行配置信道

对prime+probe和flush+reload都可用的：

-i 规定每个比特发送的时间间隔

-r 规定一个通信的缓存集

-a 规定发送方访问cache花费的周期

对于prime+probe使用：-p规定接收者填充缓存花费的周期

-b计算给定配置下的信道带宽

对于flush+reload使用：-f规定一个共享文件（不能是空文件）

**运行：**

1.首先，设置大页面，禁用超线程

./setup.sh

2.以聊天的模式运行sender和receiver.使sender和receiver在一个socket中，由于n2机器中只有一个socket，所以可随意选择核。

taskset -c 0 ./sender

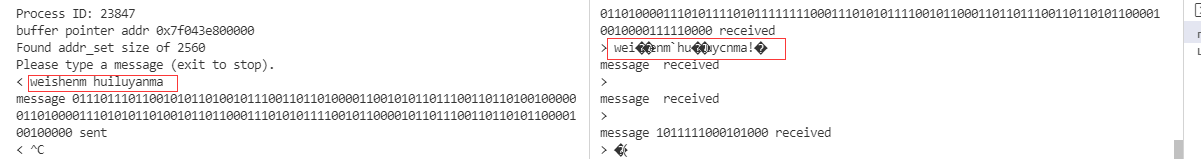
taskset -c 1 ./receiver

3.以不同的配置运行评估信道的带宽。

./benchmark.py

**运行结果：**

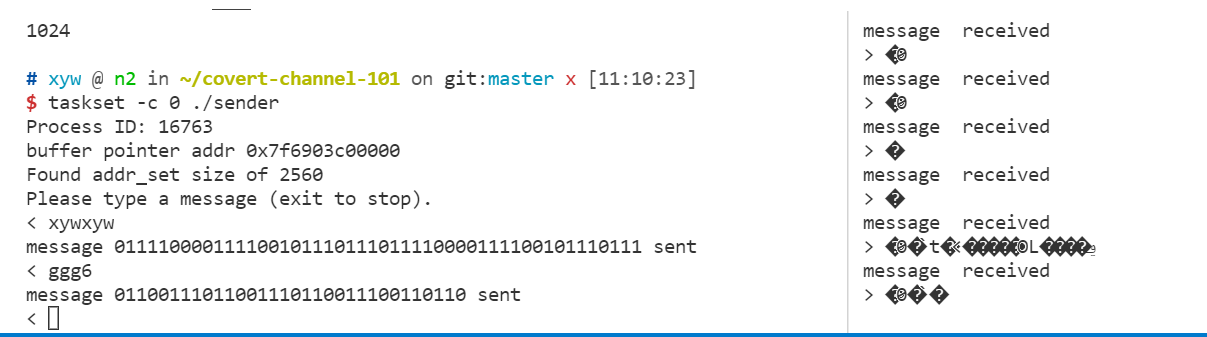
会乱码



运行多次之后发现多数情况下消息的尾部会有一个菱形问号的乱码

少数情况下消息缺失，错误，极少情况下消息完全相同，没有多余乱码符号。





Flushreload信道下不会出现乱码

**运行python benchmark.py**，结果如下：

/home/xyw/covert-channel-101

['taskset', '-c', '2', '/home/xyw/covert-channel-101/pp-llc-send', '-b']

==== Test parameters:{'interval': 2000000, 'primeTime': 800000, 'accessTime': 800000} ====

run #0...Done

Printing bit transition matrix as:

[[0 -> 0, 0 -> 1],

[1 -> 0, 0 -> 1]]

[[2352. 1701.]

[1583. 2556.]]

Capacity: 0.028465404543036074

Bandwidth: 38.52564764344833

==== Test parameters:{'interval': 1000000, 'primeTime': 400000, 'accessTime': 400000} ====

run #0...Done

Printing bit transition matrix as:

[[0 -> 0, 0 -> 1],

[1 -> 0, 0 -> 1]]

[[1319. 2750.]

[1049. 3074.]]

Capacity: 0.0042734850951846835

Bandwidth: 10.535547375388807

(py38)

**Sender.c**

**init\_config()**

**1.config->channel==PrimeProbe**

**计算并分配一个LLC大小的缓冲区（）使用mmap分配buffer得到指针config->buffer，初始化缓冲区使之成为非零页面。**

**构建addr\_set（缓存集索引为0）**

**将物理地址空间的行随机**

**2.config->channel==** **L1DPrimeProbe**

**计算并分配一个两倍L1大小的缓冲区，收缩探测集防止自驱逐。**

**3.config->channel==** **FlushReload**

**使用MMAP得到config->buffer**

**void benchmark\_send(struct config \*config\_p)**

**FILE \*senderSave = fopen("data/senderSave", "w+");**

**创建随机信息，写入文件中。**

**send\_bit\_fr(bool one, const struct config \*config)**

如果传入的one是true，则访问config->addr\_set->addr，计算时间差，如果小于interval，clflush(addr)。

否则如果时间差小于Interval，则while一直空循环。

**send\_bit\_pp(bool one, const struct config \*config)**

如果传入的one为true,while()空循环等待接收者填充缓存行

访问：

Do while()

等待接收者probe

**Main函数**

初始化config

While(sending)

send\_bit = send\_bit\_pp或者send\_bit\_fr

获得输入的字符fgets(text\_buf)，转化为二进制msg。

同步？

调用send\_bit(one,config)，如果msg[i]==1,传入one为true。

**Receiver.c**

**init\_config()**

**1. config->channel == PrimeProbe || config->channel == L1DPrimeProbe**

**其中的代码与sender.c中的L1DPrimeProbe情况下代码一样。（为什么呢，明明这里需要包括L3的情况）**

**2. config->channel == FlushReload**

**与sender.c相同。**

**detect\_bit\_fr(const struct config \*config)**

如果访问时间超过miss\_threshold个时钟周期，且小于1000（如果超过1000个周期，则为磁盘未命中，排除该情况）则未命中misses++。

循环使得sender有时间flush 缓存，如果未命中占一半，则返回true

**detect\_bit\_pp(const struct config \*config)**

**\* Detects a bit by repeatedly measuring the access time of the addresses in the probing set and counting the number of misses for the clock length of config->interval**

**通过测量对探测集地址的访问时间和计未命中次数，不断地检测一个bit**

**If the the first\_bit argument is true, relax the strict definition of "one" and try to cc\_sync with the sender.**

**如果第一个比特数据是对的，则放松one的限制定义，尝试和sender同步。**

**Prime**

**等待发送方访问。**

**Probe**

**While()**

**问题：1.debug函数怎么用？**

**2.** **#define LOG\_CACHE\_SETS\_L3 15 等这些值代表什么**

**3. 为什么Construct the addr\_set by taking the addresses that have cache set index 0**

**为什么There will be at least one of such addresses in our buffer**

**在我们的缓冲区中至少有一个这样（？）的地址，这样是指缓存集索引为0的地址吗。**

**4.** **L3 is a more restrict set是什么意思呢**

**5.** **// restrict the probing set to CACHE\_WAYS\_L1 to aviod self eviction**

**//压缩探测集至L1 way，避免自驱逐**

**if (addr\_set\_size >= 2 \* (CACHE\_WAYS\_L1 + CACHE\_WAYS\_L2)) break;**

**这个注释和代码，对得上吗？**

**6.** **send\_bit\_pp中gettime()与stoptime的关系。**

**stopTime = start\_t + config->prime\_period + config->access\_period**

**7.比如config->interval，是在哪里赋值的？**

**config->interval = atoi(optarg);**

**optarg在getopt.h中，**

**getopt函数是命令行参数解析函数**

**默认值是0x000f0000**

**8.在哪里输入参数选择channel呢？在**./sender之后吗

9. relax the strict definition of "one" and try to \* cc\_sync with the sender之后，同步操作在哪儿?

10. ImportError: No module named 'numpy'

运行pip install numpy提醒：

You are using pip version 8.1.1, however version 21.2.3 is available.

You should consider upgrading via the 'pip install --upgrade pip' command

运行pip install --upgrade pip还是提示要更新。

然后运行curl [https://bootstrap.pypa.io/get-pip.py -o get-pip.py](https://bootstrap.pypa.io/get-pip.py%20-o%20get-pip.py)

再sudo python3 get-pip.py --force-reinstall

提示ERROR: This script does not work on Python 3.5 The minimum supported Python version is 3.6. Please use https://bootstrap.pypa.io/pip/3.5/get-pip.py instead.

安装conda，

打开python3.8

conda activate py38

**选择信道**：使用-c加

PrimeProbe = 0,末尾有乱码

FlushReload=1可以完全还原发送内容且在发送端exit,接收端可同时退出

L1DPrimeProbe=2

L1DPrimeProbe信道接收不到信息，相同核不同核都不行。

**如taskset -c 0 ./sender -c 0**

**taskset -c 1 ./receiver -c 0**

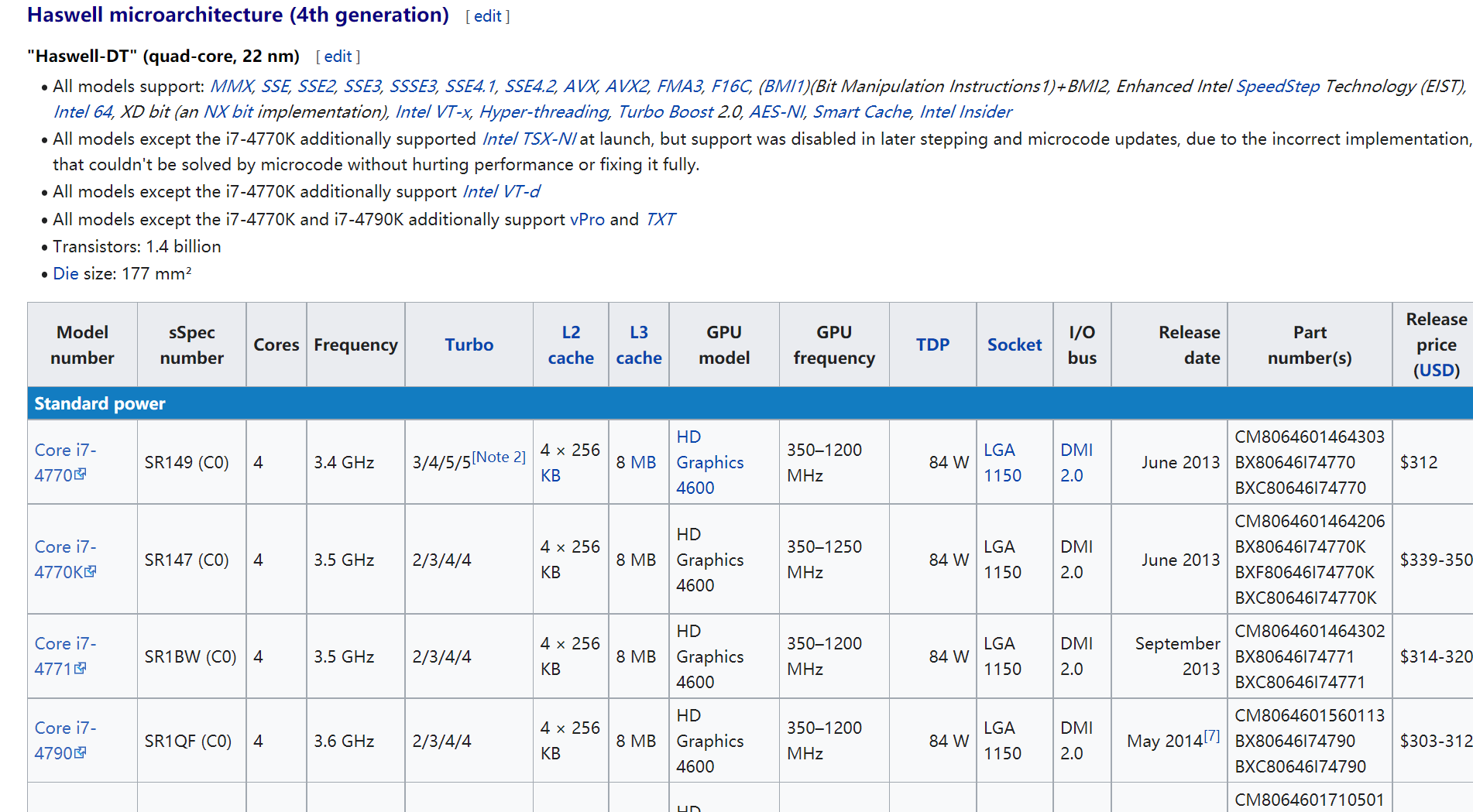
问题1.addr\_set（驱逐集）是怎么来的

问题2：gdb调试命令 gef插件

问题3：

* a local i7 Haswell (4c/8t) desktop的地址表示（是4核8线程的意思吗）
* 用手册找，在E:\读研\小组\2021.1.4\_指令
* 查看addr\_set的值

[List of Intel Core i7 processors - Wikipedia](https://en.wikipedia.org/wiki/List_of_Intel_Core_i7_processors)



The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme

Edition are based on the Haswell-E microarchitecture and support Intel 64 architecture.

怎么看地址的表示呢？

地址分布address fields

I3-9100为

### [Coffee Lake微架构](https://zh.wikipedia.org/wiki/Coffee_Lake%E5%BE%AE%E6%9E%B6%E6%9E%84)

问题4：缓存的关联性？防止自驱逐

问题5：benchmark\_send在哪儿设置的

if (config.benchmark\_mode) {

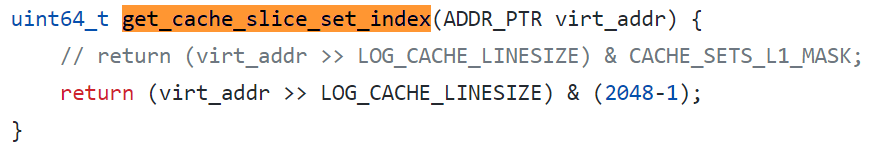
benchmark\_send(&config)；

exit(0);

问题6：比较一下之前的代码

之前的wlmnzf/PrimeProbe构建冲突集，再构建驱逐集，再得到其他缓存集的驱逐集。

而现在的代码就根据get\_cache\_slice\_set\_index(addr) == config->cache\_region来确定缓存集



问题7：找网卡的驱逐集（环缓冲区）

问题8：发送方的buffer pointer addr与接收方不同，代表什么呢

为什么这里的addr是48（二进制）位，0x7fab65600000（0x7f开头），是忽略了前面的0吗。

addr\_set中的addr是60（二进制）位(uint64\_t)？该如何打印？

匹配符是%lu吗，addr:140441642205184（140开头），改为 0x%lx之后结果如下：（缓存集索引为0就是指后8位为0）

addr 0x7f303**89**20000（389即0011 1000 1001）

addr 0x7f3039720000（397即0011 1001 0111）

addr 0x7f3026520000（265即0010 0110 0101）

addr 0x7f3027320000（273即0010 0111 0011）

addr 0x7f3028120000（281即0010 1000 0001）

addr 0x7f3028f20000（28f即0010 1000 1111）

addr 0x7f3029d20000（29d即0010 1001 1101）

addr 0x7f302ab20000(2ab即0010 1010 1011)

addr 0x7f302b920000(2b9即0010 1011 1001)

addr 0x7f302c720000(2c7即0010 1100 0111)

addr 0x7f302d520000(2d5 0010 1011 0101)

addr 0x7f302e320000(2e3 0010 1110 0011)

究竟有多少位不同，才能形成2560个地址呢（为什么发送方是固定的2560，接收方则是固定的80）

2021、9、6

0x7f2866c00000 0110 0110 1100

0x7f2869a00000 0110 1001 1010

0x7f2868c00000 0110 1000 1100

0x7f2877800000 0111 0111 1000

、、、、、、、、、、、、、、、、

address.txt

0x7fd4c5a00000 1100 0101 1010 0000 0000 0000 0000 0000

0x7fd4c4c00000 1100 0100 1100 0000

0x7fd4d4c20000 1101 0100 1010 0010

0x7fd4c8ca0000 1100 1000 1100 1010

现在观察有12位不同，从第17位（最右是第零位）

固定一个set\_index，查看地址个数和关系

答: 固定set\_index为0之后

Sender:Found addr\_set size of 160

Receiver:Found addr\_set size of 80

0x7fa9d5a00000 0x7fa9d6800000 0x7fa9d7600000 0x7fa9d8400000 0x7fa9d9200000 0x7fa9da000000 0x7fa9dae00000 0x7fa9dbc00000 0x7fa9dca00000 0x7fa9dd800000 0x7fa9de600000 0x7fa9df400000 0x7fa9e0200000 0x7fa9e1000000 0x7fa9e1e00000 0x7fa9e2c00000 0x7fa9e3a00000 0x7fa9e4800000 0x7fa9e5600000 0x7fa9e6400000 0x7fa9e7200000 0x7fa9d4000000 0x7fa9d4e00000 0x7fa9d5c00000 0x7fa9d6a00000 0x7fa9d7800000 0x7fa9d8600000 0x7fa9d9400000 0x7fa9da200000 0x7fa9db000000等等

Broardwell架构slice怎么决定的

Cloudlab

Xeon E5-2630 v3