问题：两台相同配置的不同厂商的sta，接入相同的流量，其中一台的剩余内存比另外一台少200~300M。

查询问题N步走：

1. 比较/proc/meminfo

对比两台设备的/proc/meminfo信息：10.222.3.18的用户态进程使用了2187M；10.222.3.49用户态进程1897M进程，相差了300M。所以问题出现在用户态。（这里需要说明的是 Active + Inactive就是用户态使用内存的综合）

```
TS3.0.24.4474 ~ # cat /proc/meminfo | grep -i active
Active:          1767116 kB
Inactive:         473376 kB
Active(anon):    1745808 kB
Inactive(anon):   412100 kB
Active(file):      21308 kB
Inactive(file):    61276 kB
```

10.222.3.18

```
TS3.0.24.4474 ~ # cat /proc/meminfo | grep -i active
Active:          1513568 kB
Inactive:         429908 kB
Active(anon):    1451248 kB
Inactive(anon):   360332 kB
Active(file):      62320 kB
Inactive(file):    69576 kB
```

10.222.3.49

2. 比较RSS

虽然说rss并不能够完全准确的代表进程使用的物理内存大小，但是能够看出些端倪，我们能够比较相同进程的占用来确定是否有进程占用差异很大。通过ps获取rss跟进程的对应关系，比较占用较大的一些进程；可以发现fwlog和logsd相加起来10.222.3.18要比10.222.3.49多出约250M。

这里锁定了fwlog来查看。

3. fwlog对比

依然是对比，这次我们对比两个进程的内存空间，通过pmap -x pid的方式来对比，结果如下。

可以看到10.222.3.18比10.222.3.49多使用了约140M的匿名内存页，而且这些匿名内存页都是Dirty的状态，也就是说这些内存页是正在被使用的，根据fwlog的功能分析，它不应该长时间持有

这么大的内存，说明出现了内存泄漏。

Address	Kbytes	RSS	Dirty	Mode	Mapping
0000003697816000	4	4	4	rw---	libnsl.so.1
0000003697817000	8	0	0	rw---	[anon]
00007f6380000000	440	156	156	rw---	[anon]
00007f638006e000	65096	0	0	-----	[anon]
00007f6388000000	132	12	12	rw---	[anon]
00007f6388021000	65404	0	0	-----	[anon]
00007f638c000000	41128	26936	26936	rw---	[anon]
00007f638e82a000	24408	0	0	-----	[anon]
00007f6390000000	15792	11940	11940	rw---	[anon]
00007f6390f6c000	49744	0	0	-----	[anon]
00007f63941ca000	6192	4764	4764	rw---	[anon]
00007f63947d6000	4	0	0	-----	[anon]
00007f63947d7000	8192	24	24	rw---	[anon]
00007f6394fd7000	4	0	0	-----	[anon]
00007f6394fd8000	8192	36	36	rw---	[anon]
00007f63957d8000	4	0	0	-----	[anon]
00007f63957d9000	8192	16	16	rw---	[anon]
00007f6395fd9000	4	0	0	-----	[anon]
00007f6395fda000	8192	28	28	rw---	[anon]
00007f63967da000	8	0	0	r----	ips.mo
00007f63967dc000	4	4	4	rw---	[anon]

10.222.3.18

0000003697817000	8	0	0	rw---	[anon]
00007f21ac000000	16908	8892	8892	rw---	[anon]
00007f21ad083000	48628	0	0	-----	[anon]
00007f21b4000000	132	4	4	rw---	[anon]
00007f21b4021000	65404	0	0	-----	[anon]
00007f21b8000000	65532	65532	65532	rw---	[anon]
00007f21bbfff000	4	0	0	-----	[anon]
00007f21bc000000	440	164	164	rw---	[anon]
00007f21bc06e000	65096	0	0	-----	[anon]
00007f21c0000000	132	4	4	rw---	[anon]
00007f21c0021000	65404	0	0	-----	[anon]
00007f21c4000000	57276	45476	45476	rw---	[anon]
00007f21c77ef000	8260	0	0	-----	[anon]
00007f21c8000000	65536	65472	65472	rw---	[anon]
00007f21cc000000	132	12	12	rw---	[anon]
00007f21cc021000	65404	0	0	-----	[anon]
00007f21d163a000	4104	4104	4104	rw---	[anon]
00007f21d1a3c000	4	0	0	-----	[anon]
00007f21d1a3d000	8192	32	32	rw---	[anon]
00007f21d223d000	4	0	0	-----	[anon]
00007f21d223e000	8192	16	16	rw---	[anon]
00007f21d2a3e000	4	0	0	-----	[anon]
00007f21d2a3f000	10244	560	560	rw---	[anon]
00007f21d3440000	4	0	0	-----	[anon]
00007f21d3441000	8192	24	24	rw---	[anon]
00007f21d3c41000	4	0	0	-----	[anon]
00007f21d3c42000	8192	40	40	rw---	[anon]
00007f21d4442000	4	0	0	-----	[anon]
00007f21d4443000	8192	20	20	rw---	[anon]
00007f21d4c43000	4	0	0	-----	[anon]
00007f21d4c44000	8192	28	28	rw---	[anon]

10.222.3.49

3. 分析内存泄露的内容(咨询了关大师如何查看)

我们可以用gdb来dump出想要的进程内存，例0x7f21b8000000这一块，通过如下方式：gdb: dump memory /fwlog/test_memory.dump 0x7f21b8000000 0x7F21BBFFF000

```
TS3.0.24.4474 ~ # vi test_dump.gdb
dump memory /fwlog/test_memory.dump 0x7f21b8000000 0x7F21BBFFF000
q
```

通过hexdump来查看dump出来的内存：可以看到里面存的是

session相关的字符串，而且是json格式的字符串。fwlog中只有一个地方用到了session的json转字符串。

4. 分析fwlog代码

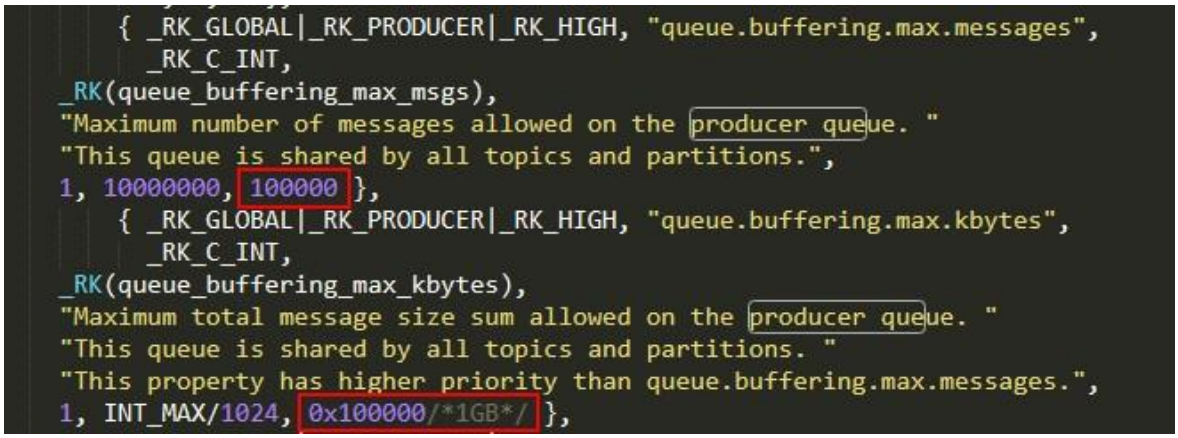
在fwlog/fwlog/write_to_thrid.cc中的write_to_third函数中。初次分析好像没有什么问题， json_object_put会释放掉json_str；深入看了以后还是证明这个json库是没有问题的，虽然写的很复杂。

在查看json库无果了以后，开始怀疑kafka发送那里可能拷贝一份json_str，因为它调用的是asyn接口，一般来说是异步的，也就是说在kafka处理前，这个json_str已经走完了释放逻辑。在kafka的实现中可以看到，它进行了produce以后就退出了，而且参数明确表示它拷贝了payload

```
// json转字符串
json_str = json_object_to_json_string(json);
if (json_str == NULL) {
    json_object_put(json);
    return -1;
}

//std::cout<<"json_str: "<<json_str<<std::endl;

if (tpc.m_type == KAFKA_TYPE) {
    //提取topic
    static Kafka_Producer s_kafka(tpc.broker_list, tpc.common_topic, tpc.compression_method);

    if (tpc.sole_topic.find(type) != tpc.sole_topic.end()) {
        std::string tmp_topic = tpc.sole_topic[type];
        if (!tmp_topic.empty()) {
            s_kafka.set_topic(tmp_topic);
        }
        else {
            s_kafka.set_topic(tpc.common_topic);
        }
    }

    // 异步发送
    if (s_kafka.async_send(json_str, strlen(json_str)) < 0) {
        json_object_put(json);
        printf("kafka send error, error: %s\n", s_kafka.get_errstr());
        return -1;
    }
}

json_object_put(json);

return 0;
```

```
int Kafka_Producer::async_send(const char *payload, unsigned int len) {
    return send(m_topic, m_partition, payload, len, 0);
}

int Kafka_Producer::sync_send(const char *payload, unsigned int len, int timeout_ms) {
    return send(m_topic, m_partition, payload, len, timeout_ms);
}

int Kafka_Producer::send(const std::string &topic, int partition, const char *payload, \
                         unsigned int len, int timeout_ms) {
    if (!m_enable) {
        return -1;
    }

    if ((m_producer == NULL) || (payload == NULL) || (len == 0)) {
        return -1;
    }

    RdKafka::ErrorCode resp = m_producer->produce(topic, partition,
                              RdKafka::Producer::RK_MSG_COPY /* Copy payload */,
                              /* Value */
                              (void *)payload, len,
                              /* Key */
                              NULL, 0,
                              /* Timestamp (defaults to now) */
                              0,
                              /* Message headers, if any */
                              NULL,
                              /* Per-message opaque value passed to
                              * delivery report */
                              NULL);
    if (resp != RdKafka::ERR_NO_ERROR) {
        m_errstr = RdKafka::err2str(resp);
        return -1;
    }

    // 同步
    if (timeout_ms) {
        resp = m_producer->flush(timeout_ms);
        if (resp != RdKafka::ERR_NO_ERROR) {
            m_errstr = RdKafka::err2str(resp);
            return -1;
        }
    }

    return 0;
}
```

至此，问题已经基本明确，当kafka服务端异常时，客户端就可能出现msg的积累，积累的数量我们也可以在代码中找到。限制是10w条和1G，按照我们的msg大小来算，最多能够累计约200M。



结论：

除了fwlog进程，还有logsd和auditd都使用了kafka，这三个进程合起来比另外一台主机上多使用了300M内存。所以所谓的内存泄漏的原因是由于kafka服务端连不上导致，但是我们在编码过程中应该识别到这种风险，毕竟内存的增长可能导致OOM，在10.222.3.18上就出现了OOM杀进程的问题。最后讨论的结果是通过配置限制kafka缓存的消息数和消息大小，来避免出现占用过多内存的问题。