内核 NAT 性能优化分析

编 写 人：韦康 佘磊 孙家昌

审 核 人：施鹤远

批 准 人：

批准日期：20 年 月 日

目录

修改控制表

序号	修改章节	修订类型	版本	修改内容	修改人	修改日期	审批人	审批日期

注： “修订类型”为“首版”、“添加”、“删除”和“修改”四种类型。

一、综述

针对 linux-2.6.33 内核 NAT 转发延时过大的问题，我方分析了内核 NAT 处理的主要流程。分析表明，内核实现 NAT 转发关键部分在于内核 Netfilter 框架。Netfilter 作为一个通用的处理框架提供了包过滤，NAT 转换等功能，但因其过多的功能模块，使得内核处理包延时增加。因此我方通过对 Netfilter 配置进行精简，减小了内核 NAT 处理时延，取得了双核百兆 NAT 不丢包的效果。

精简效果：

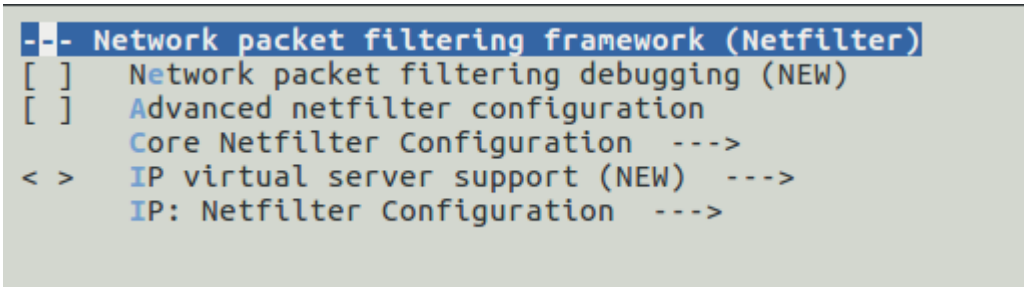
	100Mbps 丢包率	临界不丢包点
	54%(单核)	30Mbps(单核)
	51.3%(双核)	55Mbps(双核)
	11.4%(单核)	87Mbps(单核)
	不丢(双核)	>=100Mbps(双核)

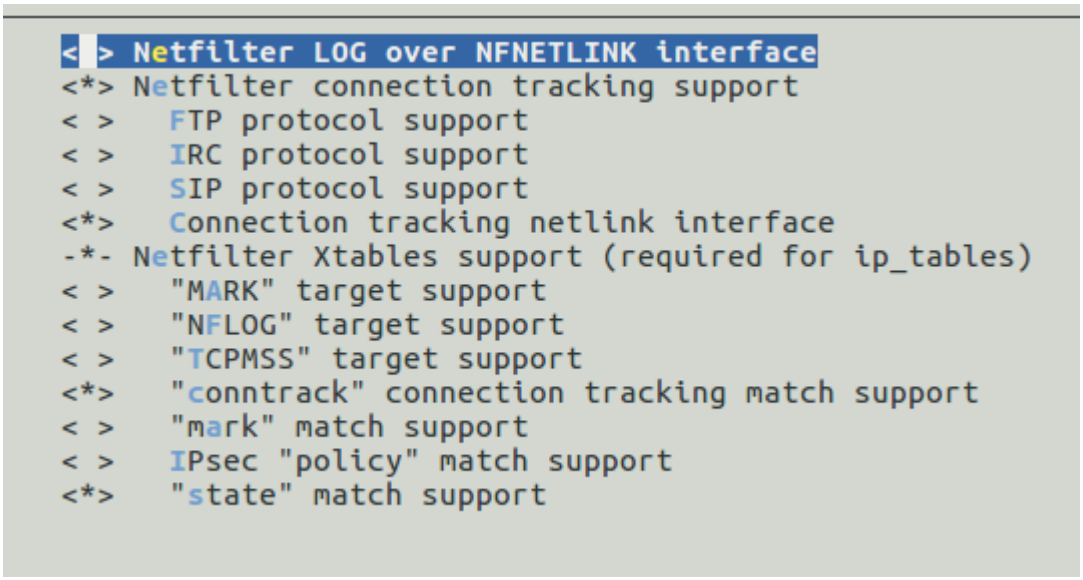
实验配置：

1. 实验采用的内核为 RT-linux-2.6.33
2. 实验采用 UDP 报文，包长为 64byte
3. 测试仪器为 IXIA
4. 实验环境所限，链路上限为 100Mbps
5. 测试功能为单次 NAT 转发

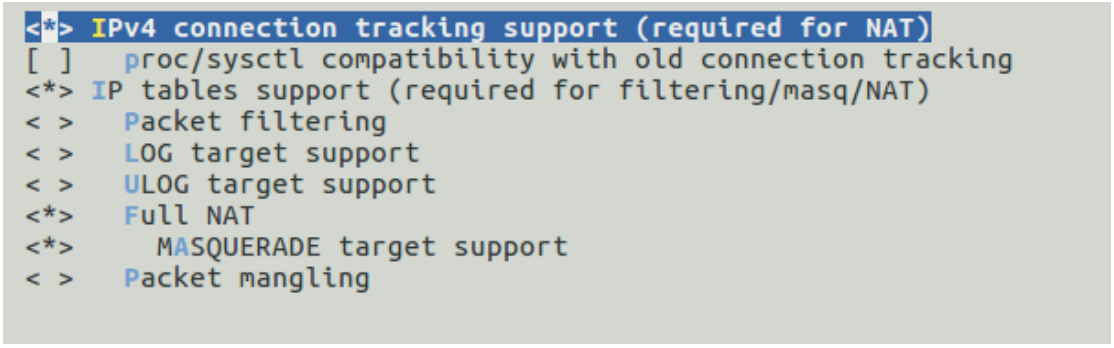
精简步骤：

Netfilter 编译选项位于 Networking-->Networking Options 下的 Network Packet Filtering Framework。可通过如下图示进行配置：





注：Core Netfilter Configuration 中配置



注：IP：Netfilter Configuration 中配置

二、详细分析

2.1 定位 Netfilter

NAT 延时产生可分为两部分：网络包通过协议栈的时间，网络包在 Netfilter 中停留的时间。完全利用 linux 内核做网络包转发时，网络包通过协议栈的时间是不可避免的，主要不确定点在于网络包在 Netfilter 中的停留时间。我们做了下列对比试验来分析 Netfilter 对转发性能的影响：

配置	包长(byte)	100Mbps 丢包情况
开启 netfilter（默认配置）	64	丢 54%
关闭 netfilter	64	不丢

注：

1. 开启 Netfilter 时采用的是默认配置，并且做 NAT 转发

2. 对比实验采用的硬件平台是 powerpc 单核

上述对比实验表明，在关闭 Netfilter 的情况下，内核只做网络包转发是可以达到百兆不丢包的，因此 Netfilter 是我们一个重要的优化点。

2.2 Netfilter 机制与钩子函数

将问题定位到 Netfilter 之后，我方对 Netfilter 的实现进行了详细分析。Netfilter 是 linux 内核中一种扩展性较强的处理框架，其在网络包的处理流程中定义了 5 个挂载点，每个挂载点上可注册一组钩子函数来实现相应的功能，如包过滤等功能。Netfilter 功能的实现是由 Netfilter 框架(提供挂载点)与钩子函数实体共同实现的。

为明确基于 Netfilter 框架实现百兆 NAT 转发可行性，我方通过实验确定了内核 NAT 处理性能上界。实验保留 Netfilter 框架本身，但在配置中去除了所有的功能模块，即各个挂载点上没有钩子函数。

配置	包长(byte)	100Mbps 丢包情况
开启 netfilter(无钩子函数)	64	不丢
开启 netfilter(默认配置)	64	丢 54%
关闭 netfilter	64	不丢

通过上述对比实验，我们可以发现在百兆情况下 Netfilter 框架本身并不会产生较大的性能损耗，真正的性能损耗来自于注册在 Netfilter 中注册的钩子函数。因此通过删减各个挂载点上不必要的钩子函数是有可能达到百兆做 NAT 转发不丢包的。

2.3 Netfilter 中 Nat 实现

在明确通过删减不必要的钩子函数来实现百兆 NAT 转发不丢包的基础上，优化任务关键在于最大限度删减不必要的钩子函数。由于 Netfilter 各个模块功能存在耦合，因此必须明确 Netfilter 中 NAT 的具体实现。我方对 Netfilter 中与 NAT 相关的重要功能模块（iptables、conntrack）做出如下分析：

iptables 为规则表。Iptables 表中存在一张 NAT 表，用户可以通过 iptables 命令可以将 NAT 转化规则写入到该 NAT 表中，netfilter 在做 NAT 转换时便会查询这张表以完成对 ip 转化，因此 iptables 机制必须保留。

conntrack 是连接跟踪模块，该模块的主要作用是记录网络包的连接状态，内核基于连接跟踪实现 NAT，因此 Conntrack 模块必须保留。

三、实验分析

在具体分析过 Netfilter 中 NAT 实现之后，我们对配置做了尽可能的精简，减少了不必要的功能和钩子函数。

	100Mbps 丢包率	临界不丢	CPU 使用率
	54%(单核)	30Mbps(单核)	87%(单核)
	51.3%(双核)	55Mbps(双核)	50%(双核)
	11.4%(单核)	87Mbps(单核)	100%(单核)
	不丢(双核)	\	50%(双核)

通过实验数据我们可以发现：在双核的情况下，通过精简配置是可以作到百兆 NAT 转发不丢包的。此外在单核时，我们发现 100Mbps 在精简配置的情况下也存在丢包的可能，不过此时 cpu 的使用率接近 100%。另外双核情况下，cpu 的使用率总体稳定在 50%左右，这与实验采用的半载网卡驱动有关。

四、自主 NAT 转换

除了精简配置之外，我们还尝试采用基于 Netfilter 框架中注册钩子函数自主实现 NAT 转换，这样做的目的是可以进一步的对 Netfilter 中的配置进行精简（如删去 Iptables 模块）。目前仅实现了静态无端口概念的 NAT 转换。

自主 NAT 转换实现流程：

在 PreRouting 挂载点注册钩子实现 DNAT

截取数据包→目的 IP 匹配→目的 IP 修改→校验和修改

在 PostRouting 挂载点注册钩子实现 SNAT

截取数据包→源 IP 匹配→源 IP 修改→校验和修改

我方对自主实现的 NAT 转发进行了性能测试，在单核百兆的情况下，基于 Netfilter 自主实现的 NAT 转发仍然存在丢包，且性能与精简配置情况下并无明

显提升。

此外，由于自主实现的NAT转发其规则是写死在代码中的，扩展性很差。并且由于自主实现的钩子函数未涉及到端口的概念，实用性有限。综合考虑，精简Netfilter配置是成本较低，效率较高的优化手段。