

**SWE-P30-02**

**iproute2之TC命令集成到busybox说明**

**V1.00**

**修 订 记 录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 修订内容简述 | 修订人 | 修订日期 | 修订后版本号 |
| 1 | 创建文档 | 钟武强 | 2013-9-13 | V1.00 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**目录**

[**1.** **简介** 4](#_Toc367095005)

[**1.1.** **目的** 4](#_Toc367095006)

[**2.** **编译tc可执行文件说明** 4](#_Toc367095007)

[**3.** **将tc移植到busybox中说明** 7](#_Toc367095008)

[**4.** **总结** 10](#_Toc367095009)

[**5.** **TC工具使用相关说明** 10](#_Toc367095010)

[**5.1.** **配置网卡的根流控节点为HTB** 10](#_Toc367095011)

[**5.2.** **建立分类树** 10](#_Toc367095012)

[**5.3.** **数据包分类** 10](#_Toc367095013)

[**5.4.** **设置每个叶子节点的流控方法** 11](#_Toc367095014)

[**6.** **参考资料清单** 11](#_Toc367095015)

[**7.** **TC脚本源码** 11](#_Toc367095016)

关键词：*iproute2 tc*

摘 要：介绍了iproute2之tc命令移植说明

缩略语清单：*对本文所用缩略语进行说明，要求提供每个缩略语的英文全名和中文解释。*

| 缩略语 | 英文全名 | 中文解释 |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. **简介**
   1. **目的**

TC是集成到iproute2中的一个功能强大的工具，可以控制网络数据收发策略。

linux内核中提供了流量控制的相关处理功能，相关代码在net/sched目录下；而应用层上的控制是通过iproute2软件包中的tc来实现，tc和sched的关系就好象iptables和netfilter的关系一样，一个是用户层接口，一个是具体实现，关于tc的使用方法可详将Linux Advanced Routing HOWTO，本文主要介绍TC工具移植的过程。

本文也详细介绍了对于一个开源工具是如何集成到busybox中的。

1. **编译tc可执行文件说明**

下载源码iproute2-2.6.29-1.tar.bz2，解压后tar –jxvf iproute2-2.6.29-1.tar.bz2，

1. iproute2-2.6.29-1/Makefile
2. 修改交叉编译器

CC = gcc

HOSTCC = gcc

改为

CC = arm-hisiv200-linux-gcc

HOSTCC = arm-hisiv200-linux-gcc

STRIP = arm-hisiv200-linux-strip

1. 采用静态编译

增加LDFLAGS += -static

1. 选择和TC功能相关的源码路径编译

修改

SUBDIRS=lib ip tc misc netem genl

改为

SUBDIRS=lib tc

1. 为减小编译后的可执行文件大小做下strip处理

修改

all: Config

@set -e; \

for i in $(SUBDIRS); \

do $(MAKE) $(MFLAGS) -C $$i; done

改为

all: Config

@set -e; \

for i in $(SUBDIRS); \

do $(MAKE) $(MFLAGS) -C $$i; done

$(STRIP) tc/tc

1. iproute2-2.6.29-1/tc/Makefile

屏蔽动态编译选项

修改

LDFLAGS += -Wl,-export-dynamic

改为

# LDFLAGS += -Wl,-export-dynamic

到目前位置，已经完成了交叉编译环境的编译了，回到iproute2-2.6.29-1/目录，执行make，能在iproute2-2.6.29-1/tc/ 目录下找到tc，这就是tc可执行程序的移植过程，还是比较简单的。

但是接下来你会发现，tc命令在建立流量控制策略的时候 报错了，如下：

./tc qdisc add dev eth0 root handle 1: htb default 5

Unknown qdisc "htb", hence option "default" is unparsable

这个是由于iproute2源码是不支持静态编译的，在执行策略的时候，会到动态库中查找策略，由于我们用的都是静态库，所以才会出现错误，

修改方法：手动初始化所有策略链表。如下：

在tc.c中添加如下代码。

extern struct qdisc\_util cbq\_qdisc\_util;

extern struct qdisc\_util drr\_qdisc\_util;

extern struct qdisc\_util dsmark\_qdisc\_util;

extern struct qdisc\_util bfifo\_qdisc\_util;

extern struct qdisc\_util pfifo\_qdisc\_util;

extern struct qdisc\_util pfifo\_fast\_qdisc\_util;

extern struct qdisc\_util gred\_qdisc\_util;

extern struct qdisc\_util htb\_qdisc\_util;

extern struct qdisc\_util htb2\_qdisc\_util;

extern struct qdisc\_util ingress\_qdisc\_util;

extern struct qdisc\_util multiq\_qdisc\_util;

extern struct qdisc\_util netem\_qdisc\_util;

extern struct qdisc\_util prio\_qdisc\_util;

extern struct qdisc\_util red\_qdisc\_util;

extern struct qdisc\_util rr\_qdisc\_util;

extern struct qdisc\_util sfq\_qdisc\_util;

extern struct qdisc\_util tbf\_qdisc\_util;

void init\_qdisc\_list(void)

{

qdisc\_list = &cbq\_qdisc\_util;

cbq\_qdisc\_util.next = &drr\_qdisc\_util;

drr\_qdisc\_util.next = &dsmark\_qdisc\_util;

dsmark\_qdisc\_util.next = &bfifo\_qdisc\_util;

bfifo\_qdisc\_util.next = &pfifo\_qdisc\_util;

pfifo\_qdisc\_util.next = &pfifo\_fast\_qdisc\_util;

pfifo\_fast\_qdisc\_util.next = &gred\_qdisc\_util;

gred\_qdisc\_util.next = &htb\_qdisc\_util;

htb\_qdisc\_util.next = &htb2\_qdisc\_util;

htb2\_qdisc\_util.next = &ingress\_qdisc\_util;

ingress\_qdisc\_util.next = &multiq\_qdisc\_util;

multiq\_qdisc\_util.next = &netem\_qdisc\_util;

netem\_qdisc\_util.next = &prio\_qdisc\_util;

prio\_qdisc\_util.next = &red\_qdisc\_util;

red\_qdisc\_util.next = &rr\_qdisc\_util;

rr\_qdisc\_util.next = &sfq\_qdisc\_util;

sfq\_qdisc\_util.next = &tbf\_qdisc\_util;

}

extern struct filter\_util basic\_filter\_util;

extern struct filter\_util flow\_filter\_util;

extern struct filter\_util fw\_filter\_util;

extern struct filter\_util route\_filter\_util;

extern struct filter\_util rsvp\_filter\_util;

extern struct filter\_util rsvp6\_filter\_util;

extern struct filter\_util tcindex\_filter\_util;

extern struct filter\_util u32\_filter\_util;

void init\_filter\_list(void)

{

filter\_list = &basic\_filter\_util;

basic\_filter\_util.next = &flow\_filter\_util;

flow\_filter\_util.next = &fw\_filter\_util;

fw\_filter\_util.next = &route\_filter\_util;

route\_filter\_util.next = &rsvp\_filter\_util;

rsvp\_filter\_util.next = &rsvp6\_filter\_util;

rsvp6\_filter\_util.next = &tcindex\_filter\_util;

tcindex\_filter\_util.next = &u32\_filter\_util;

}

extern struct action\_util \* action\_list;

extern struct action\_util gact\_action\_util;

extern struct action\_util ipt\_action\_util;

extern struct action\_util mirred\_action\_util;

extern struct action\_util nat\_action\_util;

extern struct action\_util pedit\_action\_util;

extern struct action\_util skbedit\_action\_util;

void init\_action\_list(void)

{

action\_list = &gact\_action\_util;

gact\_action\_util.next = &ipt\_action\_util;

ipt\_action\_util.next = &mirred\_action\_util;

mirred\_action\_util.next = &nat\_action\_util;

nat\_action\_util.next = &pedit\_action\_util;

pedit\_action\_util.next = &skbedit\_action\_util;

}

extern struct ematch\_util \*ematch\_list;

extern struct ematch\_util cmp\_ematch\_util;

extern struct ematch\_util meta\_ematch\_util;

extern struct ematch\_util nbyte\_ematch\_util;

extern struct ematch\_util u32\_ematch\_util;

void init\_ematch\_list(void)

{

ematch\_list = &cmp\_ematch\_util;

cmp\_ematch\_util.next = &meta\_ematch\_util;

cmp\_ematch\_util.next = &nbyte\_ematch\_util;

nbyte\_ematch\_util.next = &u32\_ematch\_util;

}

extern struct m\_pedit\_util \*pedit\_list;

extern struct m\_pedit\_util p\_pedit\_icmp;

extern struct m\_pedit\_util p\_pedit\_ip;

extern struct m\_pedit\_util p\_pedit\_ip6;

extern struct m\_pedit\_util p\_pedit\_tcp;

extern struct m\_pedit\_util p\_pedit\_udp;

void init\_pedit\_list(void)

{

pedit\_list = &p\_pedit\_icmp;

p\_pedit\_icmp.next = &p\_pedit\_ip;

p\_pedit\_ip.next = &p\_pedit\_ip6;

p\_pedit\_ip6.next = &p\_pedit\_tcp;

p\_pedit\_tcp.next = &p\_pedit\_udp;

}

tc.c的main函数中添加

init\_qdisc\_list();

init\_filter\_list();

init\_action\_list();

init\_ematch\_list();

init\_pedit\_list();

最后包含头文件

#include "m\_ematch.h"

#include "m\_pedit.h"

重新编译后tc工具就能正常使用了。

1. **将tc移植到busybox中说明**

思路是将所有tc相关的代码都编译成libtc.a库，然后在busybox中以库的方式添加进去。

第一步、要做的就是找到所有tc的相关源码，这个简单，先执行make，然后在iproute2-2.6.29-1/目录下，执行以下命令，找到所有的 .o 文件

find ./ -name \*.o

./lib/libnetlink.o

./lib/dnet\_pton.o

./lib/ll\_addr.o

./lib/ll\_proto.o

./lib/rt\_names.o

./lib/inet\_proto.o

./lib/ll\_types.o

./lib/dnet\_ntop.o

./lib/ipx\_pton.o

./lib/utils.o

./lib/ll\_map.o

./lib/ipx\_ntop.o

./tc/f\_u32.o

./tc/q\_hfsc.o

./tc/p\_tcp.o

./tc/m\_estimator.o

./tc/m\_pedit.o

./tc/q\_rr.o

./tc/f\_rsvp.o

./tc/f\_basic.o

./tc/em\_u32.o

./tc/emp\_ematch.lex.o

./tc/f\_flow.o

./tc/q\_gred.o

./tc/q\_fifo.o

./tc/tc\_util.o

./tc/tc\_estimator.o

./tc/tc\_core.o

./tc/q\_red.o

./tc/em\_cmp.o

./tc/m\_skbedit.o

./tc/m\_ematch.o

./tc/m\_nat.o

./tc/tc.o

./tc/tc\_filter.o

./tc/f\_fw.o

./tc/q\_prio.o

./tc/p\_icmp.o

./tc/tc\_red.o

./tc/q\_netem.o

./tc/f\_route.o

./tc/f\_tcindex.o

./tc/emp\_ematch.yacc.o

./tc/q\_ingress.o

./tc/q\_dsmark.o

./tc/q\_drr.o

./tc/m\_gact.o

./tc/em\_nbyte.o

./tc/tc\_monitor.o

./tc/m\_police.o

./tc/m\_mirred.o

./tc/q\_cbq.o

./tc/tc\_class.o

./tc/tc\_stab.o

./tc/q\_multiq.o

./tc/m\_action.o

./tc/q\_tbf.o

./tc/m\_ipt.o

./tc/em\_meta.o

./tc/q\_sfq.o

./tc/tc\_qdisc.o

./tc/p\_ip.o

./tc/tc\_cbq.o

./tc/q\_htb.o

./tc/p\_udp.o

以上是所有的.o文件。

第二步、在前面的基础上修改iproute2-2.6.29-1/tc/Makefile，将上述所有.o文件都编译成libtc.a

1. 增加iproute2-2.6.29-1/lib目录下的所有.o

LIBLIB :=

LIBLIB += ../lib/utils.o

LIBLIB += ../lib/rt\_names.o

LIBLIB += ../lib/ll\_types.o

LIBLIB += ../lib/ll\_proto.o

LIBLIB += ../lib/ll\_addr.o

LIBLIB += ../lib/inet\_proto.o

LIBLIB += ../lib/dnet\_ntop.o

LIBLIB += ../lib/dnet\_pton.o

LIBLIB += ../lib/ipx\_ntop.o

LIBLIB += ../lib/ipx\_pton.o

LIBLIB += ../lib/ll\_map.o

LIBLIB += ../lib/libnetlink.o

1. 修改

libtc.a: $(TCLIB)

$(AR) rcs $@ $(TCLIB)

改为

libtc.a: $(TCLIB) $(TCOBJ) $(LIBLIB)

$(AR) rcs $@ $(TCLIB) $(TCOBJ) $(LIBLIB)

第三步、由于需要编译成libtc.a所以需要修改原先的main函数名，

修改

iproute2-2.6.29-1/tc/tc.c中的int main(int argc, char \*\*argv)

改为

int dvr\_tc\_main(int argc, char \*\*argv)

第四步、回到iproute2-2.6.29-1/ 目录，执行make

这个时候会报错，是因为没有了main函数，没关系，这个时候我们需要的是iproute2-2.6.29-1/tc/libtc.a

第五步、拷贝libtc.a到busybox-1.18.4/dvrutils/hi35xx/

PS:由于busybox中的networking下已经含有tc但是功能不全，没法正常使用，为了不和原有的冲突，将新移植的tc改名为itc。

拷贝libtc.a后更名为libitc.a

第六步、在dvrutils中建立itc.c文件

拷贝以下内容

#include <stdio.h>

int dvr\_tc\_main(int argc, char \*\*argv);

int itc\_main(int argc, char \*\*argv);

int itc\_main(int argc, char \*\*argv)

{

return dvr\_tc\_main(argc, argv);

}

第7步、修改busybox-1.18.4/Makefile.flags

找到

PLATFORM = HI35XX

修改

LDLIBS += dvr cdrw fdisk hd

改为

LDLIBS += dvr cdrw fdisk hd itc dl m

其中itc是我们上面编的libitc.a ，dl 和m一个是动态库，一个是数学库，包含dlopen等，包含rint等，不加上会编译不过。

第8步、修改busybox-1.18.4/inlcude/applet.src.h

添加（按字母顺序）

IF\_ITC(APPLET(itc, \_BB\_DIR\_SBIN, \_BB\_SUID\_DROP))

第9步、修改busybox-1.18.4/include/usage.src.h

添加

#define itc\_trivial\_usage "none"

#define itc\_full\_usage "none"

第10步、修改busybox-1.18.4/dvrutils/Kbuild.src文件添加

lib-$(CONFIG\_ITC) += itc.o

第11步、修改busybox-1.18.4/dvrutils/Config.src文件添加配置项（在DVR Utilities下面）

config ITC

bool " itc"

depends on UTIL\_DH

default n

help

The itc use to network.

最后、第一次编译Busybox的时候，在Trunk/src/busybox-1.18.4目录下还没有.config。

在Trunk/src/busybox-1.18.4/dvrutils/configs目录下存放有的配置文件。

先在Trunk/build目录下执行make plat=hi35xx product=ATM，会从Trunk/src/busybox-1.18.4/dvrutils/configs下拷贝一份configs文件到Trunk/src/busybox-1.18.4目录下（当然也可以手动拷贝一份过去）。

然后在Trunk/src/busybox-1.18.4目录下执行make menucofig对Busybox进行相关配置，最后用make在Trunk/bin目录下生成busybox。用make install在Trunk/src/busybox-1.18.4/\_install目录下生成相应的文件系统。

这里make的时候还是会报错，原因是由于，移植过来的libitc.a中有函数名和networking中的重名了，处理方法是，在iproute2源码中重命名重名的函数名，全部加上前缀dh\_即可

改完后重新生成libtc.a拷贝到busybox-1.18.4/dvrutils/hi35xx下更名为libitc.a，重新执行make即可。

最后TC脚本是需要和内核配套使用的。在make menuconfig中配置如下：

Networking support --->

Networking options --->

QoS and/or fair queueing --->

全选上吧

1. **总结**

在移植过程中，碰到的最大问题是，初次接触开源源码移植，简单修改交叉编译器后，碰到执行命令报错的时候，不知道从何入手查问题，对于动态库的概念理解的不够深，总结一点，还是需要多查资料，Google是很好的工具。

1. **TC工具使用相关说明**

HTB将各种类别的流控处理节点组合成一个节点树, 每个叶节点是一个流控结构, 可在叶子节点使用不同的流控方法，如将pfifo, tbf等。HTB一个重要的特点是能设置每种类型的基本带宽，当本类带宽满而其他类型带宽空闲时可以向其他类型借带宽。注意这个树是静态的, 一旦TC命令配置好后就不变了, 而具体的实现是HASH表实现的, 只是逻辑上是树, 而且不是二叉树, 每个节点可以有多个子节点。

  HTB运行过程中会将不同类别不同优先权的数据包进行有序排列，用到了有序表, 其数据结构实际是一种特殊的二叉树, 称为红黑树(Red Black Tree), 这种树的结构是动态变化的，而且数量不只一个，最大可有8×8个树。

现以简单分类配置为说明。

* 1. **配置网卡的根流控节点为HTB**

#根节点ID是0x10000, 缺省类别是0x10012,

# handle x:y, x定义的是类别ID的高16位, y定义低16位

#注意命令中的ID参数都被理解为16进制的数

tc qdisc add dev eth0 root handle 1: htb default 12

#在内核中将调用htb\_init()函数初始化HTB流控结构.

* 1. **建立分类树**

#根节点总流量带宽100kbps, 内部类别ID是0x10001

tc class add dev eth0 parent 1: classid 1:1 htb rate 100kbps ceil 100kbps

#第一类别数据分30kbps, 最大可用100kbps, 内部类别ID是0x10010(注意这里确实是16进制的10)

tc class add dev eth0 parent 1:1 classid 1:10 htb rate 30kbps ceil 100kbps

#第二类别数据分30kbps, 最大可用100kbps, 内部类别ID是0x10011

tc class add dev eth0 parent 1:1 classid 1:11 htb rate 10kbps ceil 100kbps

#第三类别(缺省类别)数据分60kbps, 最大可用100kbps, 内部类别ID是0x10012

tc class add dev eth0 parent 1:1 classid 1:12 htb rate 60kbps ceil 100kbps

#在内核中将调用htb\_change\_class()函数来修改HTB参数

* 1. **数据包分类**

#对源地址为1.2.3.4, 目的端口是80的数据包为第一类, 0x10010

tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip src 1.2.3.4 match ip

dport 80 0xffff flowid 1:10

#对源地址是1.2.3.4的其他类型数据包是第2类, 0x10011

tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip src 1.2.3.4 flowid

1:11

#其他数据包将作为缺省类, 0x10012

#在内核中将调用htb\_find\_tcf(), htb\_bind\_filter()函数来将为HTB绑定过滤表

* 1. **设置每个叶子节点的流控方法**

# 1:10节点为pfifo

tc qdisc add dev eth0 parent 1:10 handle 20: pfifo limit 10

# 1:11节点也为pfifo

tc qdisc add dev eth0 parent 1:11 handle 30: pfifo limit 10

# 1:12节点使用sfq, 扰动时间10秒

tc qdisc add dev eth0 parent 1:12 handle 40: sfq perturb 10

#在内核中会使用htb\_leaf()查找HTB叶子节点, 使用htb\_graft()函数来设置叶子节点的流控方法.

1. **参考资料清单**

《说明文档\_网络限速-HTB介绍及使用方法.docx》

《经验案例\_网络限速TC命令修改\_已解决.docx》

1. **TC脚本源码**

#!/bin/sh

devName=eth0

maxBand="the maxband is"

defaultrate=$1

#给出个最大上限带宽100M

maxtrate=102400

sourceip=$2

#sourceport=$2

a=4

let ratearg=$defaultrate/$a

clear()

{

tc qdisc del dev $devName root handle 1:

}

addroot()

{

tc qdisc add dev $devName root handle 1: $1 default $2

}

addclass()

{

parentid=$1

classid=$2

type=$3

maxspeed=$4

if [ $# -ne 5 ]; then

tc class add dev $devName parent 1:$parentid classid 1:$classid $type rate ${maxspeed}kbit

else

tc class add dev $devName parent 1:$parentid classid 1:$classid $type rate ${ratearg}kbit ceil ${maxspeed}kbit prio $5

fi

}

addqdisc()

{

parrentid=$1

handdleid=$2

type=$3

tc qdisc add dev $devName parent 1:$parrentid handle $handdleid:0 $type perturb 10

}

clear

if [ $defaultrate != 0 ]; then

echo $maxBand $defaultrate

addroot htb 5

addclass 0 1 htb $maxtrate

addclass 1 5 htb $maxtrate 2

addclass 1 10 htb $defaultrate 10

#源IP限速

tc filter add dev $devName protocol ip parent 1:0 prio 1 u32 match ip dst $sourceip flowid 1:10

#port限速

# tc filter add dev $devName protocol ip parent 1:0 prio 1 u32 match ip sport 80 0xffff flowid 1:10

else

echo "No Limit"

fi

#addqdisc 5 5 sfq

#addqdisc 10 10 sfq

#tc filter add dev $devName protocol ip parent 1:0 prio 1 u32 match ip dport $sourceport 0xffff flowid 1:5