Qualcomm Technologies, Inc.

Linux UDP Speed Test

Application Note

80-Y7706-40 Rev. AA

April 21, 2022

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
AA	April 2022	Initial release

Note: There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

Contents

1 Introduction 5

1.1 Purpose 5
1.2 UDP speed test 5

2 UDP speed test user space application 6

2.1 Commands 6
2.1.1 init 6
2.1.2 final 6
2.1.3 create 6
2.1.4 list/clear 7
2.1.5 start/stop 7
2.1.6 stats 7
2.2 Parameter values 8
2.3 Command execution examples 8

3 UDP speed test pseudo driver 10

3.1 Data structures 10
3.1.1 Global structures 10
3.1.2 Database for configuration rules 11
3.1.3 Result statistics 11
3.1.4 File operations for `/dev/nss_udp_st` 12
3.2 APIs 12
3.2.1 nss_udp_st_init 12
3.2.2 nss_udp_st_write 12
3.2.3 nss_udp_st_ioctl 13
3.2.4 nss_udp_st_read 13
3.2.5 nss_udp_st_exit 13
3.2.6 nss_udp_st_tx 13
3.2.7 nss_udp_st_rx_ip<v4/v6>_pre_routing_hook 14

4 VLAN and PPPoE support 15

4.1 APIs 15
4.1.1 nss_udp_st_tx 15
4.2 Testing 15

5 Test topology 16

5.1 Tx side 16
5.2 Rx side 17

A References..... 18
A.1 Acronyms and terms.....18

Figures

Figure 1-1 UDP speed test block diagram.....5

1 Introduction

1.1 Purpose

This document defines the API for the Linux (host data path) based User Datagram Protocol (UDP) speed test. A Linux kernel module (LKLM), e.g., pseudo driver, has been implemented to generate and transmit UDP packets on the Tx path and count incoming packets using a net filter prerouting hook on the Rx path.

The user space application `nss_udp_st` receives input parameters and starts/stops the speed test. Then, the LKLM (pseudo driver) calls `nss_udp_st.ko` to perform the speed test on the transmit and receive paths. For the transmit path, this module would generate and send X number of UDP packets per time quanta based on parameters received from the user space application. For the receive path, the module would implement a prerouting hook to receive and count packets on the path.

The application and pseudo driver will communicate using traditional ioctl on the character device `/dev/nss_udp_st`.

1.2 UDP speed test

The UDP speed test measures downlink and uplink performance of the device. In the transmit direction, we allocate buffers, encapsulate them with IP and UDP headers, and send those buffers out of the device to measure transmit speed of the device.

In receive direction, we have buffers to measure the receive speed of the device. A 5-tuple check is done in the receive direction.

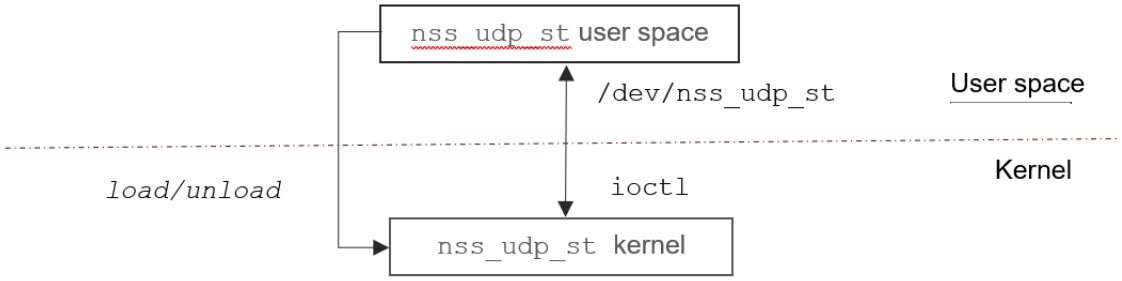


Figure 1-1 UDP speed test block diagram

2 UDP speed test user space application

The `nss_udp_st` user space application supports the UDP speed test. The application takes in command line arguments in a specific format to pass as test parameters.

The application and pseudo driver will communicate using traditional ioctl on the character device `/dev/nss_udp_st`.

This chapter describes the supported commands.

2.1 Commands

2.1.1 init

Loads pseudo driver and passes parameters for rate, buffer size, and dscp to the driver. This command also creates a directory for rules and stats.

```
nss_udp_st --mode=<init> --rate=<rate> --buffer_sz=<buffer_size> --
dscp=<value> -net_dev=<net_dev>
```

The application creates the following files for stats and rules:

```
/tmp/nss-udp-st/rules
/tmp/nss-udp-st/tx_stats
/tmp/nss-udp-st/rx_stats
```

Sends `NSS_UDP_ST_IOCTL_INIT` to `/dev/nss_udp_st`.

2.1.2 final

Unloads pseudo driver. This command is called at the end of the test.

```
nss_udp_st --mode=<final>
```

NOTE: Stats/rules are not cleared.

2.1.3 create

Adds a configuration rule to the database. This command is called after `init`.

```
nss_udp_st --mode=<create> --sip=<sip> --dip=<dip> --sport=<sport> --
dport=<dport> --version=<version>
```

The application maintains a database of rules (5-tuple information) at some configurable location in the file system (for example, `/tmp/nss-udp-st/rules`). Creating a rule would add a row to this database.

2.1.4 list/clear

Lists or deletes all configured rules from the database.

```
nss_udp_st --mode=<list/clear>
```

The application lists or clears all rules from the database maintained at configurable location (for example, `/tmp/nss-udp-st/rules`).

2.1.5 start/stop

Starts or stops Tx/Rx speed test using configured test parameters (rules).

```
nss_udp_st --mode=<start> --type=<tx/rx> --time=<time>
nss_udp_st --mode=<stop>
```

Time is an optional parameter indicating the duration of the test. If not specified, the test stops after a default timeout (4 minutes). A test can also be stopped using the `stop` command.

2.1.5.1 start

- Test direction (Tx/Rx) is stored in the pseudo driver.
- Reads rules database from configurable location `/tmp/nss-udp-st/rules` and writes each rule to `/dev/nss_udp_st`.
- Upon receiving a success return code after reading/writing, command sends `NSS_UDP_ST_IOCTL_START` to `/dev/nss_udp_st` along with the optional time parameter.

2.1.5.2 stop

Sends `NSS_UDP_ST_IOCTL_STOP` code to `/dev/nss_udp_st`.

2.1.6 stats

Gets Rx/Tx stats for the speed test.

```
nss_udp_st --mode=stats type=<tx/rx>
cat /tmp/nss-udp-st/<rx/tx>_stats
```

- Stats are “read” from the pseudo driver and streamed to a configurable location: `/tmp/nss-udp-st/tx_stats` or `/tmp/nss-udp-st/rx_stats`.
- If the pseudo driver is loaded, the application would “read” test results from `/dev/nss_udp_st`. The driver read handler copies the current state of the stats struct to the user buffer passed. The app then formats received data in XML and copies it into configurable file `/tmp/nss-udp-st/<tx_stats/rx_stats>`.
  - On success, the number of bytes read (= size of stats struct) is returned from the driver.
  - On failure, an error code is returned from the driver (e.g., if stats are checked before the test starts).

- For every `mode=stats` command, stats for that test *type (tx/rx)* will be overwritten in the `/tmp/nss-udp-st/<tx/rx>_stats` file.

2.1.6.1 Example XML format for Tx/Rx stats

<tx stats>

```
<start>jiffies</start>
<packets>count</packets>
<bytes>count</bytes>
<end ms-per-jiffies="xxx">jiffies</end>
```

<rx stats>

```
<start>jiffies</start>
<packets>count</packets>
<bytes>count</bytes>
<end ms-per-jiffies="xxx">jiffies</end>
```

2.2 Parameter values

Parameters	Description
mode	Start/stop/create/list/clear/stats/init/final
type	Tx/Rx
sip	Source IP address (e.g., DUT WAN IP address)
dip	Destination IP address (e.g., speed test server IP address)
sport	Source Port
dport	Destination port
version	IPv4/IPv6 protocol
net_dev	Interface name where the cable connected to (e.g., eth0, pppoe-wan, eth0.100 etc.)
time	Duration in seconds for test
rate	Expected rate in Mbps
buffer_sz	Size of UDP packet
dscp	DSCP field for the IP header

2.3 Command execution examples

The following examples show possible valid flows (not an exhaustive list) for Tx/Rx tests.

- Add new rules

  init → **create (x # of times)** → start [tx/rx] [time] → stop → stats → final

- List and use existing rules

  init → **list** → start [tx/rx] [time] → stop → stats → final

- Clear all rules and set up new rules
  init → **clear** → **create (x # of times)** → start [tx/rx] [time] → stop → stats → final
- Query stats during test
  init → create (x # of times) → start [tx/rx] [time] → **stats** → stop → stats → final

3 UDP speed test pseudo driver

The LKLM `nss_udp_st.ko` generates and transmits UDP packets based on parameters received from the user space application for the transmit path and implements a prerouting hook to receive and count packets on the receive path.

This module also manages the transmit/receive stats and passes test results to the user space application via the `/dev/nss_udp_st` char device node. The application and pseudo driver will communicate using traditional ioctl on the character device `/dev/nss_udp_st`.

3.1 Data structures

3.1.1 Global structures

```
struct nss_udp_st {
        struct nss_udp_st_param config; /* config params for tx */
        struct nss_udp_st_rules rules; /* database for config rules */
        struct nss_udp_st_stats stats; /* result statistics */
        uint32_t rule_count;          /* no of rules configured */
        uint64_t time;                /* duration of test */
        bool mode;                     /* start =0; stop=1 */
        bool dir;                      /* tx=0; rx=1 */
};

struct nss_udp_st_param {
        uint32_t rate;                    /* target rate in Mbps */
        uint32_t buffer_sz;               /* buffer size of each packet */
        uint32_t dscp;                    /* dscp flag for tx packet */
        char net_dev [NSS_UDP_ST_IFNAMSZ];     /* net device interface */
};
```

3.1.2 Database for configuration rules

The database for configuration rules is implemented as a linked list.

```
struct nss_udp_st_rules {
        struct list_head list;          /* kernel's list structure */
        struct nss_udp_st_ip sip;       /* source ip */
        struct nss_udp_st_ip dip;       /* dest ip */
        uint16_t sport;                 /* source port */
        uint16_t dport;                 /* dest port */
        uint16_t flags;                 /* version of IP address */
        uint8_t dst_mac [ETH_ALEN];     /* dest mac */
};

struct nss_udp_st_ip {
        union {
                uint32_t ipv4;          /* IPv4 address. */
                uint32_t ipv6[4];    /* IPv6 address. */
        } ip.
};
```

3.1.3 Result statistics

```
struct nss_udp_st_stats {
        struct nss_udp_st_pkt_stats p_stats;
        /* Packet statistics */
        atomic_long_t timer_stats [NSS_UDP_ST_STATS_TIME_MAX] ;
        /* Time statistics */
        atomic_long_t errors [NSS_UDP_ST_ERROR_MAX];
        /* Error statistics */
        bool first_pkt;
        /* First packet flag */
};

struct nss_udp_st_pkt_stats {
        atomic_long_t tx_packets;   /* Number of packets transmitted */
        atomic_long_t tx_bytes;     /* Number of bytes transmitted */
        atomic_long_t rx_packets;   /* Number of packets received */
        atomic_long_t rx_bytes;     /* Number of bytes received */
};

enum nss_udp_st_stats_time {
        NSS_UDP_ST_STATS_TIME_START,
        /* Start time of the test */
        NSS_UDP_ST_STATS_TIME_CURRENT,
        /* Current time of the running test */
        NSS_UDP_ST_STATS_TIME_ELAPSED,
        /* Elapsed time of the current test */
        NSS_UDP_ST_STATS_TIME_MAX
        /* Maximum timer statistics type */
};
```

```
enum nss_udp_st_error {
        NSS_UDP_ST_ERROR_NONE,
        /* no error */
        NSS_UDP_ST_ERROR_INCORRECT_RATE,
        /* incorrect rate */
        NSS_UDP_ST_ERROR_INCORRECT_BUFFER_SIZE,
        /* incorrect buffer size */
        NSS_UDP_ST_ERROR_MEMORY_FAILURE,
        /* Memory allocation failed */
        NSS_UDP_ST_ERROR_INCORRECT_IP_VERSION,
        /* Incorrect IP version */
        NSS_UDP_ST_ERROR_PACKET_DROP,
        /* Packet Drop */
        NSS_UDP_ST_ERROR_MAX
        /* Maximum error statistics type */
};
```

3.1.4 File operations for `/dev/nss_udp_st`

```
static const struct file_operations nss_udp_st_ops = {
        . open          = nss_udp_st_open,
        . read          = nss_udp_st_read,
        . write         = nss_udp_st_write,
        . unlocked_ioctl = nss_udp_st_ioctl,
        . release       = nss_udp_st_release,
};
```

3.2 APIs

3.2.1 nss_udp_st_init

Creates char device `/dev/nss_udp_st` and initializes global structs.

3.2.2 nss_udp_st_write

- Receives configuration parameters from user space application.
- Performs MAC address resolution check and validates IP address.
  - On success, add this rule to `nss_udp_st_param` list and returns number of bytes written.
  - On failure, it returns the relevant error code.

3.2.3 nss_udp_st_ioctl

3.2.3.1 NSS_UDP_ST_IOCTL_INIT

- Receives rate, buffer size, and dscp fields from the user space application.
- Initializes fields in global struct.

3.2.3.2 NSS_UDP_ST_IOCTL_START_<TX/RX>

- Unsets global *mode* flag and set *direction* flag in the global `nss_udp_st struct`.
- Schedules `nss_udp_st_tx` for Tx.
- Registers the pre_routing hook `nss_udp_st_rx_ip<v4/v6>_pre_routing_hook` for Rx.

3.2.3.3 NSS_UDP_ST_IOCTL_STOP

- Sets global *mode* flag to indicate test has stopped.
- Unregisters pre_routing hook `nss_udp_st_rx_ip<v4/v6>_pre_routing_hook` for Rx.

3.2.4 nss_udp_st_read

Copies values from global `nss_udp_st_stats struct` to user buffer.

3.2.5 nss_udp_st_exit

Cleans up resources allocated in `nss_udp_st_init`.

3.2.6 nss_udp_st_tx

- Calculates the number of packets to be transmitted per 100 ms (for the entire test duration), based on the rate received as input.
- Sets up hrtimer to generate and send required number of packets per config rule.
  - If *mode* flag indicates test has not stopped, then UDP packets are generated and transmit to a specified interface using `ndo_start_xmit()`.
  - atomic64 increments the Tx stats in the global `nss_udp_st_stats struct`.
  - Add a check for global *mode* flag to ensure test is not stopped and keeps track of timer. Timer should start only after the first packet is sent.
  - On timeout, the *mode* flag should be modified to indicate that the test has stopped.
- Returns control to user space application.

3.2.7 nss_udp_st_rx_ip<v4/v6>_pre_routing_hook

- If mode indicates test has not stopped, match 5tuple of incoming packet with the 5tuple values from the rules database to determine if it is a speed test packet.
  - If it is a speed test packet, drop it and atomic64 increments Rx stats in the global `nss_udp_st_stats struct` (return *NF_STOLEN*).
- Sends packet to the network stack (return *NF_ACCEPT*).

NOTE: Implement spinlock to access list.

4 VLAN and PPPoE support

VLAN, PPPoE, and PPPoE over VLAN interfaces are also supported on the WAN. The scenarios currently supported are:

- eth0 ←→ DUT (simple physical interface)
- eth0-eth0.10 ←→ DUT (single VLAN)
- eth0-pppoe-wan ←→ DUT (PPPoE only)
- eth0-eth0.10-pppoe-wan ←→ DUT (PPPoE over VLAN)

4.1 APIs

4.1.1 nss_udp_st_tx

- When the Tx test starts (based on the `net_dev->type`), VLAN or PPPoE header is configured and stored in a global variable.
- A global variable *xmit_dev* is used to store the physical interface which transmits the generated packets with *ndo_start_xmit*.
- VLAN/PPPoE headers are generated before populating the Ethernet header while creating the packet.

4.2 Testing

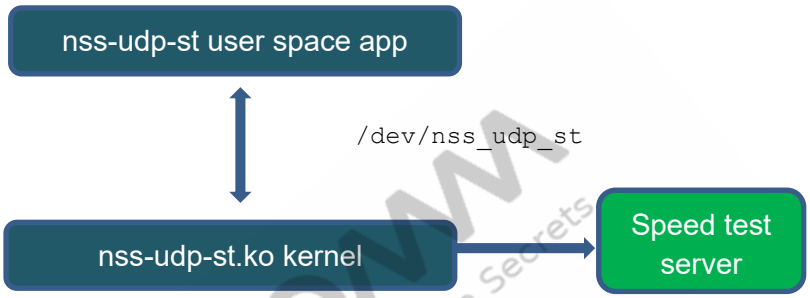
Specify the VLAN/PPPoE interface in the ***init*** command for both Tx/Rx (see flows in Section 2.3).

For example:

```
nss_udp_st --mode=<init> --rate=<rate> --buffer_sz=<buffer_size> --
dscp=<value> -net_dev=<eth0.10 or pppoe-wan>
```

5 Test topology

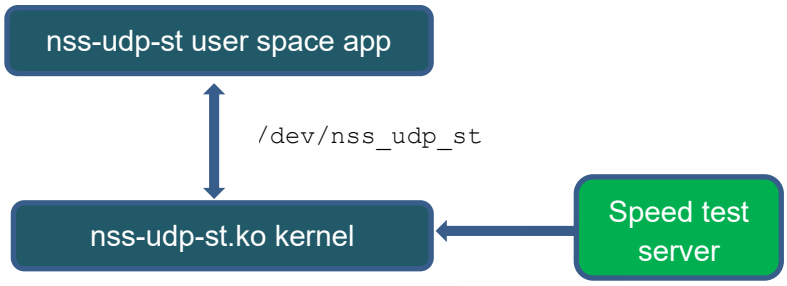
5.1 Tx side




```
nss-udp-st --mode init --rate 3000 --buffer_sz 1500 --dscp 0 -net_dev eth5
nss-udp-st --mode create --sip 192.168.2.1 --dip 192.168.2.2 --sport 1000 -
--dport 1000 --version 4
nss-udp-st --mode start --type tx
nss-udp-st --mode stats --type tx
cat /tmp/nss-udp-st/tx_stats
nss-udp-st --mode stop
nss-udp-st --mode final
```

```
root@OpenWrt:/#
root@OpenWrt:/# nss-udp-st --mode init --rate 3000 --buffer_sz 1500 --dscp 0 -ne
t_dev eth5
root@OpenWrt:/#
root@OpenWrt:/# nss-udp-st --mode create --sip 192.168.2.1 --dip 192.168.2.2 --s
port 10000 --dport 10000 --version 4
root@OpenWrt:/#
root@OpenWrt:/# ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2): 56 data bytes
64 bytes from 192.168.2.2: seq=0 ttl=64 time=1.487 ms
64 bytes from 192.168.2.2: seq=1 ttl=64 time=0.650 ms
^C
--- 192.168.2.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.650/1.068/1.487 ms
root@OpenWrt:/# nss-udp-st --mode start --type tx
root@OpenWrt:/#
root@OpenWrt:/# nss-udp-st --mode stats --type tx
root@OpenWrt:/#
root@OpenWrt:/# cat /tmp/nss-udp-st/tx_stats

Packet Stats
        tx_packets = 896310 packets
        tx_bytes   = 1357013340 bytes

Time Stats
        start time = 42950178360 ms
        capture time = 42950181810 ms
        elapsed time = 3450 ms

Error Stats
        incorrect rate = 0
        incorrect buffer size = 0
        memory failure = 0
        packet drop count = 0
        incorrect ip version = 0

Throughput Stats
        throughput   = 3146 Mbps
root@OpenWrt:/#
```

5.2 Rx side




```
nss-udp-st --mode init --rate 3000 --buffer_sz 1500 --dscp 0 -net_dev eth5
nss-udp-st --mode create --sip 192.168.2.1 --dip 192.168.2.2 --sport 1000 -
-dport 1000 --version 4
nss-udp-st --mode start --type rx
nss-udp-st --mode stats --type rx
cat /tmp/nss-udp-st/rx_stats
nss-udp-st --mode stop
nss-udp-st --mode final
```

```
root@OpenWrt:/# nss-udp-st --mode init --rate 3000 --buffer_sz 1500 --dscp 0 -ne
t_dev eth5
root@OpenWrt:/#
root@OpenWrt:/# nss-udp-st --mode create --sip 192.168.2.1 --dip 192.168.2.2 --s
port 10000 --dport 10000 --version 4
root@OpenWrt:/#
root@OpenWrt:/# nss-udp-st --mode start --type rx
root@OpenWrt:/# nss-udp-st --mode stats --type rx
root@OpenWrt:/#
root@OpenWrt:/# cat /tmp/nss-udp-st/rx_stats

Packet Stats

        rx_packets = 1969326 packets
        rx_bytes   = 2981559564 bytes

Time Stats
        start time = 42993028820 ms
        capture time = 42993038300 ms
        elapsed time = 9480 ms

Error Stats
        incorrect rate = 0
        incorrect buffer size = 0
        memory failure = 0
        packet drop count = 0
        incorrect ip version = 0

Throughput Stats
        throughput  = 2516 Mbps
root@OpenWrt:/#
root@OpenWrt:/#
```

A References

A.1 Acronyms and terms

Acronym or term	Definition
LKLM	Linux kernel module
UDP	User Datagram Protocol