NAME

cxgbetool — Userspace companion to *cxgbe*(4)

SYNOPSIS

cxgbetool *nexus command* [*parameter ...*]

cxgbetool *nexus* clearstats *port_id*
cxgbetool *nexus* clip {hold | release} *ipv6-address*
cxgbetool *nexus* clip list
cxgbetool *nexus* context {ingress | egress | fl | cong} *cntxt_id*
cxgbetool *nexus* hashfilter mode
cxgbetool *nexus* hashfilter *filter-specification*
cxgbetool *nexus* hashfilter *idx* delete
cxgbetool *nexus* hashfilter list
cxgbetool *nexus* filter mode
cxgbetool *nexus* filter *idx filter-specification*
cxgbetool *nexus* filter *idx* delete [prio {0 | 1}]
cxgbetool *nexus* filter list
cxgbetool *nexus* i2c *port_id devaddr addr* [*len*]
cxgbetool *nexus* loadcfg *fw-config.txt*
cxgbetool *nexus* loadcfg clear
cxgbetool *nexus* loadfw *fw-image.bin*
cxgbetool *nexus* memdump *addr len*
cxgbetool *nexus* policy *cop.txt*
cxgbetool *nexus* policy clear
cxgbetool *nexus* {reg | reg64} *addr*[*=val*]
cxgbetool *nexus* regdump [*register-block ...*]
cxgbetool *nexus* sched-class *sub-command* [*param value*]
cxgbetool *nexus* sched-queue *port queue class*
cxgbetool *nexus* stdio
cxgbetool *nexus* tcb *tid*

DESCRIPTION

cxgbetool provides command-line access to features and debug facilities exported by *cxgbe*(4) via private ioctls. The target nexus device, *t4nex%d* , is always the first argument. (The parent nexus for an Ethernet port *cxgbe%d* is listed in *dev.cxgbe.%d.%parent* in the *sysctl*(8) MIB). The rest consists of a command and any parameters required by that command.

Commands

clearstats *port_id*
Clear all transmit, receive, and error statistics of all queues associated with a port. The total number of ports attached to a nexus is listed in *dev.t4nex.%d.nports* and the 0 based *port_id* identifies a port within this range.

clip hold *ipv6-address*
Install a reference on the given *ipv6-address* in the CLIP (Compressed Local IPv6) table. The address is added to the CLIP table if it is not present there already.

clip list
List the contents of the CLIP table.

clip release *ipv6-address*
Release a reference on the given *ipv6-address* in the CLIP table. A reference on the address must have been acquired previously.

context ingress *ingress_cntxt_id*

`context cong` *ingress_cntxt_id*
`context egress` *egress_cntxt_id*
`context fl` *flm_cntxt_id*
Display hardware context for an ingress queue, congestion manager, egress queue, or freelist manager.

ingress_cntxt_id context id of an ingress queue -- the value listed in one of *dev.t4nex.%d.fwq.cntxt_id*, *dev.cxgbe.%d.rxq.%d.cntxt_id*, or *dev.cxgbe.%d.ofld_rxq.%d.cntxt_id*.

egress_cntxt_id context id of an egress queue -- the value listed in one of *dev.t4nex.%d.mgmtq.cntxt_id*, *dev.cxgbe.%d.txq.%d.cntxt_id*, *dev.cxgbe.%d.ctrlq.%d.cntxt_id*, *dev.cxgbe.%d.ofld_txq.%d.cntxt_id*, *dev.cxgbe.%d.rxq.%d.fl.cntxt_id*, or *dev.cxgbe.%d.ofld_rxq.%d.fl.cntxt_id*. Note that freelists are egress queues too.

flm_cntxt_id context id of a freelist manager. The FLM context id is displayed in the egress context dump of a freelist as FLMcontextID.

`hashfilter mode`
`filter mode`
Display a list of match-criteria available for use in filter rules. A full list of match-criteria known to the chip is in the table below but not all can be used together and the firmware sets up the available parameters based on "filterMode" in the configuration file. Every filter must conform to the filter mode -- multiple match criteria per filter are allowed but only from among those in the current setting of the filter mode. The filter mode for hash filters is a subset of that for normal TCAM filters and depends on the "filterMask" setting in the firmware configuration file. Hash filters do not support masked matches and an exact value for every parameter in the output of "hashfilter mode" (except ipv4/ipv6) must be provided when creating a hash filter.

(Note that *mask* defaults to all 1s when not provided explicitly. Hash filters do not support masked matches. Also note that many of the items being matched are discrete numeric values rather than bit fields and should be masked with caution.)

Criteria	Usage	Matches if ...
ipv4	`type ipv4`	incoming packet is an IPv4 datagram.
ipv6	`type ipv6`	incoming packet is an IPv6 datagram.
sip	`sip` *addr*[/*mask*]	bitwise and of the source address in an incoming IP datagram with *mask* equals *addr*. *addr* be an IPv4 or IPv6 address.
dip	`dip` *addr*[/*mask*]	bitwise and of the destination address in an incoming IP datagram with *mask* equals *addr*. *a* can be an IPv4 or IPv6 address.
sport	`sport` *port*[:*mask*]	bitwise and of the source port in an incoming TCP or UDP datagram with *mask* equals *port*.
dport	`dport` *port*[:*mask*]	bitwise and of the destination port in an incoming TCP or UDP datagram with *mask* equals *po*
fcoe	`fcoe` {0 | 1}	incoming frame is Fibre Channel over Ethernet(1) or not(0).
iport	`iport` *val*[:*mask*]	bitwise and of the ingress port with *mask* equals *val*. The ingress port is a 3 bit number that id tifies the port on which a frame arrived. Physical ports are numbered 0-3 and 4-7 are internal lo back paths within the chip. Note that ingress port is not a bit field so it is not always possibl match an arbitrary subset of ingress ports with a single filter rule.
ovlan	`ovlan` *tag*[:*mask*]	bitwise and of the 16-bit outer VLAN tag of an incoming frame with *mask* equals *tag*.
vlan	`vlan` *tag*[:*mask*]	bitwise and of the 16-bit VLAN tag of an incoming QinQ frame with *mask* equals *tag*. The i VLAN tag is used if the incoming frame is QinQ.
tos	`tos` *val*[:*mask*]	bitwise and of the 8-bit IP Type of Service/IPv6 Traffic Class in an incoming packet with *m* equals *val*.
proto	`proto` *ipproto*[:*mask*]	bitwise and of the 8-bit IP protocol in an incoming packet with *mask* equals *ipproto*.
ethtype	`ethtype` *type*[:*mask*]	bitwise and of the 16-bit Ethernet type field of an incoming frame with *mask* equals *type*.

macidx	`macidx` *idx*[:*mask*]	bitwise and of the MAC Address Match Index of an incoming frame with *mask* equals *idx*. MAC Address Match Index refers to an entry in the MPS TCAM or in the MPS hash. `matchtype` for more information.
matchtype	`matchtype` *type*[:*mask*]	bitwise and of the Match Type of an incoming frame with *mask* equals *idx*. Match Type is on the following: 0 destination MAC in incoming frame is a unicast L2 address that is programmed in the M TCAM. `macidx` can be used to match the index (and thus the MAC address) of the matc the TCAM. 1 destination MAC in incoming frame is a unicast L2 address that "hit" a hash entry in the M hash table. `macidx` can be used to match the index of the entry in the MPS hash table. 2 destination MAC in incoming frame is a multicast L2 address that is programmed in the M TCAM. `macidx` can be used to match the index (and thus the MAC address) of the matc the TCAM. 3 destination MAC in incoming frame is a multicast L2 address that "hit" an entry in the MPS h table. 4 interface on which incoming frame was received is in promiscuous mode and the destina MAC in the frame is not a broadcast address, and does not match in the MPS TCAM or the M hash either. (The frame would have been discarded if the interface wasn't in promiscu mode.) 5 interface on which incoming frame was received is in promiscuous mode and the destina MAC in the frame is not a broadcast address; it wasn't looked up in the MPS TCAM or the M hash because the chip is configured to give precedence to promiscuous mode classification. 6 destination MAC in incoming frame is a broadcast address. 7 Not documented. Do not use.
frag	`frag` {0 | 1}	incoming frame is part of a fragmented IP datagram(1) or not(0).

`hashfilter` *filter-specification*
`filter` *idx filter-specification*
Program a filter.

TCAM filters: The number of available filters is in dev.<nexus>.<instance>.nfilters. *idx* must be an unused index between 0 and nfilters - 1. IPv6 filters consume 4 consecutive entries on T4/T5 and 2 on T6 and *idx* must be aligned to 4 or 2 in this case.

Hash filters: These reside in the card's memory instead of its TCAM and are enabled with a special configuration file that is selected with `hw.cxgbe.config_file="hashfilter"` in loader.conf. There are at least half a million filters available with the sample config shipped with the driver. Note that the hardware selects the index for a hashfilter and this index is displayed when the filter is created. Hash and TCAM filters can be used together.

filter-specification consists of one or more matches (see Usage in the table above) to try against an incoming frame, an action to perform when all matches succeed, and some additional operational parameters. Hashfilters require an exact value for the 5-tuple (sip, dip, sport, dport, proto) and for any other match-criteria listed in "hashfilter mode". Possible filter actions are `drop`, `pass`, or `switch`.

Operational parameters that can be used with all filters:

`hitcnts`	Count filter hits: 0 or 1 (default).
`prio`	Filter has priority over active and server regions of TCAM: 0 (default) or 1.

Operational parameters that can be used with filters with `action pass`:

`queue`	Context id of an ingress queue to which to deliver the packet. The context id is available in *dev.cxgbe.%d.rxq.%d.cntxt_id*. By default, packets that hit a filter with action pass are delivered based on their RSS hash as usual. Use this to steer them to a particular queue.
`rpttid`	Report the filter tid instead of the RSS hash in the rx descriptor. 0 (default) or 1.

`tcbhash` Select TCB hash information in rx descriptor. 0 (default) or 1

Operational parameters that can be used with filters with `action switch`:

`eport` Egress port number on which to send the packet matching the filter. 0 to dev.<nexus>.<instance>.nports - 1.

`dmac` Replace packet destination MAC address with the one provided before switching it out of eport.

`smac` Replace packet source MAC address with the one provided before switching it out of eport.

`swapmac` Swap packet source and destination MAC addresses before switching it out of eport.

`vlan` Insert, remove, or rewrite the VLAN tag before switching the packet out of eport. `none` removes the tag, `=`*`tag`* replaces the existing tag with the one provided, and `+`*`tag`* inserts the given tag into the frame.

`nat` Specify the desired NAT mode. Valid NAT modes values are:

- `dip` Perform NAT on destination IP.
- `dip-dp` Perform NAT on destination IP, destination port.
- `dip-dp-sip` Perform NAT on destination IP, destination port, source IP.
- `dip-dp-sp` Perform NAT on destination IP, destination port, source port.
- `sip-sp` Perform NAT on source IP, source port.
- `dip-sip-sp` Perform NAT on destination IP, source IP, source port.
- `all` Perform NAT on all 4-tuple fields.

`natflag` Perform NAT only on segments which do not have TCP FIN or RST set.

`natseq` Perform NAT only if incoming segment's sequence number + payload length is less than this supplied value.

`nat_dip` Perform NAT using this destination IP.

`nat_sip` Perform NAT using this source IP.

`nat_dport` Perform NAT using this destination port.

`nat_sport` Perform NAT using this source port. Perform NAT only if incoming segment's sequence number + payload length is less than this supplied value.

`hashfilter` *`idx`* `delete`
`filter` *`idx`* `delete`
Delete filter that is at the given index.

`filter list`
List all filters programmed into the hardware.

`i2c` *`port_id devaddr addr`* [*`len`*]

`loadcfg` *`fw-config.txt`*
Install the firmware configuration file contained in *`fw-config.txt`* to the card. Set hw.cxgbe.config_file="flash" in loader.conf to get *cxgbe*(4) to use the on-flash configuration.

`loadcfg clear`
Erase configuration file from the card.

`loadfw` *`fw-image.bin`*
Install the firmware contained in *`fw-image.bin`* to the card.

`memdump` *`addr len`*
Display *`len`* bytes of data of the card's memory starting at *`addr`*. The card's memory map is available in *dev.t4nex.%d.misc.meminfo*.

`policy` *`cop.txt`*
Install the Connection Offload Policy (COP) in *`cop.txt`*. A COP offers fine-grained control over which connections get offloaded and with what parameters. Set `hw.cxgbe.cop_managed_offloading="1"` in loader.conf to ensure that t4_tom will not offload any connection before a COP is installed. Note that t4_tom must be loaded and operational (IFCAP_TOE enabled) as always for any kind of offload based on the hardware TOE.

COP installed	cop_managed_offloading	Behavior
NO	0	offload all [Default]
NO	1	no offload
YES	Don't Care	Rule based offload

The policy file consists of empty lines, comments (lines beginning with #) and any number of rules. Rules are applied in the order they appear in the file and processing stops at the first match. There is an implicit rule that disables offload for connections that do not match anything in the policy.

Each rule consists of a filter part, which determines what connections the rule applies to, and a settings part, which determines whether matching connections will be offloaded and, if so, with what settings. The general form of a rule is

`[ socket-type ] pcap-filter => settings`

socket-type is one of the following.

A
Active open. Connection is being opened by this host.

P
Passive open. Connection was requested by a peer.

L
Listen called on a socket. Disabling offload in such a rule will prevent a hardware listener from being started.

D
Don't care. Matches all of the above.

pcap-filter is an expression that follows the *pcap-filter*(7) syntax, or it is the keyword `all` that matches everything.

settings determine whether connections matching *socket-type* and *pcap-filter* are offloaded and optionally sets some per-connection properties if they are. A combination of the following is allowed.

`offload` Connection should be offloaded. Use `!offload` or `not offload` to disable offload instead.
`coalesce` Enable rx payload coalescing. Negate to disable.
`timestamp` Enable TCP timestamp option. Negate to disable.
`sack` Enable TCP Selective Acknowledgements (SACK). Negate to disable.
`nagle` Enable Nagle's algorithm. Negate to disable.
`ecn` Enable Explicit Congestion Notification (ECN). Negate to disable.
`ddp` Use Direct Data Placement (zero copy receive) and zero copy transmit on the connection to service AIO requests on the socket. Negate to disable.
`tls` Set ULP mode to ULP_MODE_TLS.
`cong` *algo* Use the specified congestion control algorithm. *algo* must be one of `reno`, `tahoe`, `newreno`, or `highspeed`.
`class` *sc* Bind the connection to the specified tx scheduling class. Valid range is 0 to 14 (for T4) and 0 to 15 (T5 onwards).
`rxq` *qnum* Use the specified offload rx queue. *qnum* should be `random`, `roundrobin`, or a number between 0 and nofldrxq for the ifnet.
`txq` *qnum* Use the specified offload tx queue. *qnum* should be `random`, `roundrobin`, or a number between 0 and nofldtxq for the ifnet.
`bind` *qnum* Shorthand for `rxq` *qnum* `txq` *qnum*. Use when nofldrxq is the same as nofldtxq.
`mss` *val* Set the advertised TCP MSS in the SYN for this connection to *val* (in bytes). The hardware MTU table must already have an entry that is suitable for the MSS.

Example of a COP.
Note that hardware listener for port 22 will be IPv4 only because the rule before it will prevent any IPv6 servers other than the first two. Also note that outgoing connections to 192.168/16 are the only outgoing connections that will get offloaded.

```
[L] port 80 => offload
[L] port 443 => offload
[L] ip6 => !offload
[L] port 22 => offload
[P] dst port 80 => offload cong highspeed !sack !ecn
[P] dst port 443 => offload tls
[A] dst net 192.168/16 => offload
[A] all => !offload
[D] port 22 => offload !nagle
```

`policy clear`
Remove the Connection Offload Policy (COP) if one is in use.

`{reg | reg64}` *`addr`*`[=`*`val`*`]`

`regdump [`*`register-block ...`*`]`
Display contents of device registers. One or more *`register-block`* can be specified to limit the registers displayed. The default is to display registers for all blocks. Registers with read side effects are not read during a `regdump` operation. *`register-block`* can be `sge pci dbg mc ma edc0 edc1 cim tp ulp_rx ulp_tx pmrx pmtx mps cplsw smb i2c mi uart pmu sf pl le ncsi xgmac`.

`sched-class config [`*`param value`*`]`
Configure optional feature capabilities for the TX scheduler.

type *`scheduler-type`*
Use packet for the packet scheduler.

minmax *`value`*
A non-zero value will enable "minmax" mode; a zero value will disable "minmax" mode.

NOTE: Many (most) of the parameters and constraints are adapter-specific - for instance the number of channels and classes which are available whether various modes are implemented, etc. Consult the adapter documentation for specific information on any limitations.

`sched-class params [`*`param value`*`]`
Configure parameters for a scheduling class.

type *`scheduler-type`*
Use packet for packet scheduler.

level *`scheduler-hierarchy-level`*
The "level" within the scheduling hierarchy which is being programed:

cl-rl Class Rate Limiting.

cl-wrr Class Weighted Round Robin.

ch-rl Channel Rate Limiting.

mode *`scheduler-mode`*
The mode in which the scheduling class is going to operate:

class All of the "flows" bound to the scheduling class will be held to aggregate scheduling constraints.

flow Each of the "flows" bound to the scheduling class will be held to the scheduling constraints.

E.g. if the scheduling class has a TX bandwidth of 10Mb/s, in `class` mode, all of the "flows" bound to the class would be limited to an aggregate bandwidth of 10Mb/s; but in `flow` mode, each of the "flows" bound to the scheduling class would be limited to 10Mb/s.

rate-unit *scheduler-rate-unit*
The units of the scheduler rate constraints:

bits bit rate in Kb/s.

pkts packets/s.

rate-mode *scheduler-rate-mode*
The mode of the scheduler rate constraints:

relative percent of port rate.

absolute Kb/s.

channel *scheduler-channel-index*
The scheduling channel to which the scheduling class will be bound.

class *scheduler-class-index*
The scheduling class being programmed.

min-rate *minimum-rate*
The minimum guaranteed rate to which a rate-limiting scheduling class hierarchy will have access.

max-rate *maximum-rate*
The maximum rate for a rate-limiting scheduling class hierarchy.

weight *round-robin-weight*
The weight to be used for a weighted-round-robin scheduling hierarchy.

pkt-size *average-packet-size*
The average packet size will be used to compute scheduler constraints for a rate-limited scheduler class hierarchy.

NOTE: Many (most) of the parameters and constraints are adapter-specific - for instance the number of channels and classes which are available, whether various modes are implemented, etc. Consult the adapter documentation for specific information on any limitations.

`sched-queue` *port queue class*
Bind the indicated port's NIC TX *queue* to the specified TX Scheduler *class*. If the TX *queue* is `all`, `*` or any negative value, the binding will apply to all of the TX queues associated with the *interface*. If the class is `unbind`, `clear` or any negative value, the TX queue(s) will be unbound from any current TX Scheduler Class binding.

`stdio`
Switch to interactive mode.

`tcb` *tid*
Display contents of the hardware TCB (TCP Control Block) for the connection identfied by *tid*.

FILES

/sys/dev/cxgbe/t4_ioctl.h

AUTHORS

This manual page was written by Navdeep Parhar <np@FreeBSD.org>.