NAME

dtrace_tcp - a DTrace provider for tracing events related to the tcp(4) protocol

SYNOPSIS

tcp:::accept-established(*pktinfo_t **, *csinfo_t **, *ipinfo_t **, *tcpsinfo_t **, *tcpinfo_t **);

tcp:::accept-refused(*pktinfo_t **, *csinfo_t **, *ipinfo_t **, *tcpsinfo_t **, *tcpinfo_t **);

tcp:::connect-established(*pktinfo_t **, *csinfo_t **, *ipinfo_t **, *tcpsinfo_t **, *tcpinfo_t **);

tcp:::connect-refused(*pktinfo_t **, *csinfo_t **, *ipinfo_t **, *tcpsinfo_t **, *tcpinfo_t **);

tcp:::connect-request(*pktinfo_t **, *csinfo_t **, *ipinfo_t **, *tcpsinfo_t **, *tcpinfo_t **);

tcp:::receive(*pktinfo_t **, *csinfo_t **, *ipinfo_t **, *tcpsinfo_t **, *tcpinfo_t **);

tcp:::send(*pktinfo_t **, *csinfo_t **, *ipinfo_t **, *tcpsinfo_t **, *tcpinfo_t **);

tcp:::state-change(*void **, *csinfo_t **, *void **, *tcpsinfo_t **, *void **, *tcplsinfo_t **);

tcp:::siftr(*siftrinfo_t **);

DESCRIPTION

The DTrace **tcp** provider allows users to trace events in the tcp(4) protocol implementation. This provider is similar to the dtrace_ip(4) and dtrace_udp(4) providers, but additionally contains probes corresponding to protocol events at a level higher than packet reception and transmission. All **tcp** probes except for **tcp:::state-change**() and **tcp:::siftr**() have the same number and type of arguments. The last three arguments are used to describe a TCP segment: the *ipinfo_t* argument exposes the version-agnostic fields of the IP header, while the *tcpinfo_t* argument exposes the TCP header, and the *tcpsinfo_t* argument describes details of the corresponding TCP connection state, if any. Their fields are described in the ARGUMENTS section.

The **tcp:::accept-established**() probe fires when a remotely-initiated active TCP open succeeds. At this point the new connection is in the ESTABLISHED state, and the probe arguments expose the headers associated with the final ACK of the three-way handshake. The **tcp:::accept-refused**() probe fires when a SYN arrives on a port without a listening socket. The probe arguments expose the headers associated with the RST to be transmitted to the remote host in response to the SYN segment.

The **tcp:::connect-established**(), **tcp:::connect-refused**(), and **tcp:::connect-request**() probes are similar to the ‘accept’ probes, except that they correspond to locally-initiated TCP connections. The

tcp:::connect-established() probe fires when the SYN-ACK segment of a three-way handshake is received from the remote host and a final ACK is prepared for transmission. This occurs immediately after the local connection state transitions from SYN-SENT to ESTABLISHED. The probe arguments describe the headers associated with the received SYN-ACK segment. The **tcp:::connect-refused**() probe fires when the local host receives a RST segment in response to a SYN segment, indicating that the remote host refused to open a connection. The probe arguments describe the IP and TCP headers associated with the received RST segment. The **tcp:::connect-request**() probe fires as the kernel prepares to transmit the initial SYN segment of a three-way handshake.

The **tcp:::send**() and **tcp:::receive**() probes fire when the host sends or receives a TCP packet, respectively. As with the dtrace_udp(4) provider, **tcp** probes fire only for packets sent by or to the local host; forwarded packets are handled in the IP layer and are only visible to the dtrace_ip(4) provider.

The **tcp:::state-change**() probe fires upon local TCP connection state transitions. Its first, third and fifth arguments are currently always NULL. Its last argument describes the from-state in the transition, and the to-state can be obtained from args[3]->tcps_state.

The **tcp:::siftr**() probe fires when a TCP segment is sent or received by the host. For a detailed description see siftr(4). The *siftrinfo_t* argument provides the information about the TCP connection.

ARGUMENTS

The *pktinfo_t* argument is currently unimplemented and is included for compatibility with other implementations of this provider. Its fields are:

uinptr_t pkt_addr	Always set to 0.

The *csinfo_t* argument is currently unimplemented and is included for compatibility with other implementations of this provider. Its fields are:

uintptr_t cs_addr	Always set to 0.
uint64_t cs_cid	A pointer to the *struct inpcb* for this packet, or NULL.
pid_t cs_pid	Always set to 0.

The *ipinfo_t* type is a version-agnostic representation of fields from an IP header. Its fields are described in the dtrace_ip(4) manual page.

The *tcpsinfo_t* type is used to provide a stable representation of TCP connection state. Some **tcp** probes, such as **tcp:::accept-refused**(), fire in a context where there is no TCP connection; this argument is

NULL in that case. Its fields are:

uintptr_t tcps_addr	The address of the corresponding TCP control block. This is currently a pointer to a *struct tcpcb*.
int tcps_local	A boolean indicating whether the connection is local to the host. Currently unimplemented and always set to -1.
int tcps_active	A boolean indicating whether the connection was initiated by the local host. Currently unimplemented and always set to -1.
uint16_t tcps_lport	Local TCP port.
uint16_t tcps_rport	Remote TCP port.
string tcps_laddr	Local address.
string tcps_raddr	Remote address.
int32_t tcps_state	Current TCP state. The valid TCP state values are given by the constants prefixed with ‘TCPS_’ in */usr/lib/dtrace/tcp.d*.
uint32_t tcps_iss	Initial send sequence number.
uint32_t tcps_suna	Initial sequence number of sent but unacknowledged data.
uint32_t tcps_snxt	Next sequence number for send.
uint32_t tcps_rack	Sequence number of received and acknowledged data.
uint32_t tcps_rnxt	Next expected sequence number for receive.
u_long tcps_swnd	TCP send window size.
int32_t tcps_snd_ws	Window scaling factor for the TCP send window.
u_long tcps_rwnd	TCP receive window size.

int32_t tcps_rcv_ws
Window scaling factor for the TCP receive window.

u_long tcps_cwnd TCP congestion window size.

u_long tcps_cwnd_ssthresh
Congestion window threshold at which slow start ends and congestion avoidance begins.

uint32_t tcps_sack_fack
Last sequence number selectively acknowledged by the receiver.

uint32_t tcps_sack_snxt
Next selectively acknowledge sequence number at which to begin retransmitting.

uint32_t tcps_rto Round-trip timeout, in milliseconds.

uint32_t tcps_mss Maximum segment size.

int tcps_retransmit A boolean indicating that the local sender is retransmitting data.

int tcps_srtt Smoothed round-trip time.

The *tcpinfo_t* type exposes the fields in a TCP segment header in host order. Its fields are:

uint16_t tcp_sport Source TCP port.

uint16_t tcp_dport Destination TCP port.

uint32_t tcp_seq Sequence number.

uint32_t tcp_ack Acknowledgement number.

uint8_t tcp_offset Data offset, in bytes.

uint8_t tcp_flags TCP flags.

uint16_t tcp_window TCP window size.

uint16_t tcp_checksum	Checksum.
uint16_t tcp_urgent	Urgent data pointer.
*struct tcphdr *tcp_hdr*	A pointer to the raw TCP header.

The *tcplsinfo_t* type is used by the **tcp:::state-change**() probe to provide the from-state of a transition. Its fields are:

int32_t tcps_state	A TCP state. The valid TCP state values are given by the constants prefixed with 'TCPS_' in */usr/lib/dtrace/tcp.d*.

The *siftrinfo_t* type is used by the **tcp:::siftr**() probe to provide the state of the TCP connection. Its fields are:

uint8_t direction	Direction of packet that triggered the log message. Either "0" for in, or "1" for out.
uint8_t ipver	The version of the IP protocol being used. Either "1" for IPv4, or "2" for IPv6.
uint16_t lport	The TCP port that the local host is communicating via.
uint16_t rport	The TCP port that the remote host is communicating via.
string laddr	The IPv4 or IPv6 address of the local host.
string raddr	The IPv4 or IPv6 address of the remote host.
uint32_t snd_cwnd	The current congestion window (CWND) for the flow, in bytes.
uint32_t snd_wnd	The current sending window for the flow, in bytes. The post scaled value is reported, except during the initial handshake (first few packets), during which time the unscaled value is reported.
uint32_t rcv_wnd	The current receive window for the flow, in bytes. The post scaled value is always reported.
uint32_t t_flags2	The current value of the t_flags2 for the flow.

uint32_t snd_ssthresh	The slow start threshold (SSTHRESH) for the flow, in bytes.
int conn_state	A TCP state. The valid TCP state values are given by the constants prefixed with 'TCPS_' in */usr/lib/dtrace/tcp.d*.
uint32_t mss	The maximum segment size (MSS) for the flow, in bytes.
uint32_t srtt	The current smoothed RTT (SRTT) for the flow in microseconds.
u_char sack_enabled	SACK enabled indicator. 1 if SACK enabled, 0 otherwise.
u_char snd_scale	The current window scaling factor for the sending window.
u_char rcv_scale	The current window scaling factor for the receiving window.
u_int t_flags	The current value of the t_flags for the flow.
uint32_t rto	The current retransmission timeout (RTO) for the flow in microseconds. Divide by HZ to get the timeout length in seconds.
u_int snd_buf_hiwater	The current size of the socket send buffer in bytes.
u_int snd_buf_cc	The current number of bytes in the socket send buffer.
u_int rcv_buf_hiwater	The current size of the socket receive buffer in bytes.
u_int rcv_buf_cc	The current number of bytes in the socket receive buffer.
u_int sent_inflight_bytes	The current number of unacknowledged bytes in-flight. Bytes acknowledged via SACK are not excluded from this count.
int t_segqlen	The current number of segments in the reassembly queue.
u_int flowid	Flowid for the connection. A caveat: Zero '0' either represents a valid flowid or a default value when the flowid is not being set.
u_int flowtype	Flow type for the connection. Flowtype defines which protocol fields are hashed to produce the flowid. A complete listing is available in */usr/include/sys/mbuf.h* under M_HASHTYPE_*.

FILES

/usr/lib/dtrace/tcp.d	DTrace type and translator definitions for all the probes of the **tcp** provider except the **siftr** probe.
/usr/lib/dtrace/siftr.d	DTrace type and translator definitions for the **siftr** probe of the **tcp** provider.

EXAMPLES

The following script logs TCP segments in real time:

```
#pragma D option quiet
#pragma D option switchrate=10hz

dtrace:::BEGIN
{
        printf(" %3s %15s:%-5s      %15s:%-5s %6s  %s\n", "CPU",
            "LADDR", "LPORT", "RADDR", "RPORT", "BYTES", "FLAGS");
}

tcp:::send
{
        this->length = args[2]->ip_plength - args[4]->tcp_offset;
        printf(" %3d %16s:%-5d -> %16s:%-5d %6d  (", cpu, args[2]->ip_saddr,
            args[4]->tcp_sport, args[2]->ip_daddr, args[4]->tcp_dport,
            this->length);
        printf("%s", args[4]->tcp_flags & TH_FIN ? "FIN|" : "");
        printf("%s", args[4]->tcp_flags & TH_SYN ? "SYN|" : "");
        printf("%s", args[4]->tcp_flags & TH_RST ? "RST|" : "");
        printf("%s", args[4]->tcp_flags & TH_PUSH ? "PUSH|" : "");
        printf("%s", args[4]->tcp_flags & TH_ACK ? "ACK|" : "");
        printf("%s", args[4]->tcp_flags & TH_URG ? "URG|" : "");
        printf("%s", args[4]->tcp_flags == 0 ? "null " : "");
        printf("\b)\n");
}

tcp:::receive
{
        this->length = args[2]->ip_plength - args[4]->tcp_offset;
        printf(" %3d %16s:%-5d <- %16s:%-5d %6d  (", cpu,
            args[2]->ip_daddr, args[4]->tcp_dport, args[2]->ip_saddr,
            args[4]->tcp_sport, this->length);
        printf("%s", args[4]->tcp_flags & TH_FIN ? "FIN|" : "");
```

```
            printf("%s", args[4]->tcp_flags & TH_SYN ? "SYN|" : "");
            printf("%s", args[4]->tcp_flags & TH_RST ? "RST|" : "");
            printf("%s", args[4]->tcp_flags & TH_PUSH ? "PUSH|" : "");
            printf("%s", args[4]->tcp_flags & TH_ACK ? "ACK|" : "");
            printf("%s", args[4]->tcp_flags & TH_URG ? "URG|" : "");
            printf("%s", args[4]->tcp_flags == 0 ? "null " : "");
            printf("\b)\n");
    }
```

The following script logs TCP connection state changes as they occur:

```
    #pragma D option quiet
    #pragma D option switchrate=25hz

    int last[int];

    dtrace:::BEGIN
    {
            printf("   %12s %-20s    %-20s %s\n",
                "DELTA(us)", "OLD", "NEW", "TIMESTAMP");
    }

    tcp:::state-change
    {
            this->elapsed = (timestamp - last[args[1]->cs_cid]) / 1000;
            printf("   %12d %-20s -> %-20s %d\n", this->elapsed,
                tcp_state_string[args[5]->tcps_state],
                tcp_state_string[args[3]->tcps_state], timestamp);
            last[args[1]->cs_cid] = timestamp;
    }

    tcp:::state-change
    /last[args[1]->cs_cid] == 0/
    {
            printf("   %12s %-20s -> %-20s %d\n", "-",
                tcp_state_string[args[5]->tcps_state],
                tcp_state_string[args[3]->tcps_state], timestamp);
            last[args[1]->cs_cid] = timestamp;
    }
```

The following script uses the siftr probe to show the current value of CWND and SSTHRESH when a packet is sent or received:

```
#pragma D option quiet
#pragma D option switchrate=10hz

dtrace:::BEGIN
{
        printf(" %3s %16s:%-5s %16s:%-5s %10s %10s\n",
            "DIR", "LADDR", "LPORT", "RADDR", "RPORT", "CWND", "SSTHRESH");
}

tcp:::siftr
{
        printf(" %3s %16s:%-5d %16s:%-5d %10u %10u\n",
            siftr_dir_string[args[0]->direction],
            args[0]->laddr, args[0]->lport, args[0]->raddr, args[0]->rport,
            args[0]->snd_cwnd, args[0]->snd_ssthresh);
}
```

COMPATIBILITY

This provider is compatible with the **tcp** provider in Solaris.

SEE ALSO

dtrace(1), dtrace_ip(4), dtrace_sctp(4), dtrace_udp(4), dtrace_udplite(4), siftr(4), tcp(4), SDT(9)

HISTORY

The **tcp** provider first appeared in FreeBSD 10.0.

AUTHORS

This manual page was written by Mark Johnston <*markj@FreeBSD.org*>.

BUGS

The *tcps_local* and *tcps_active* fields of *tcpsinfo_t* are not filled in by the translator.