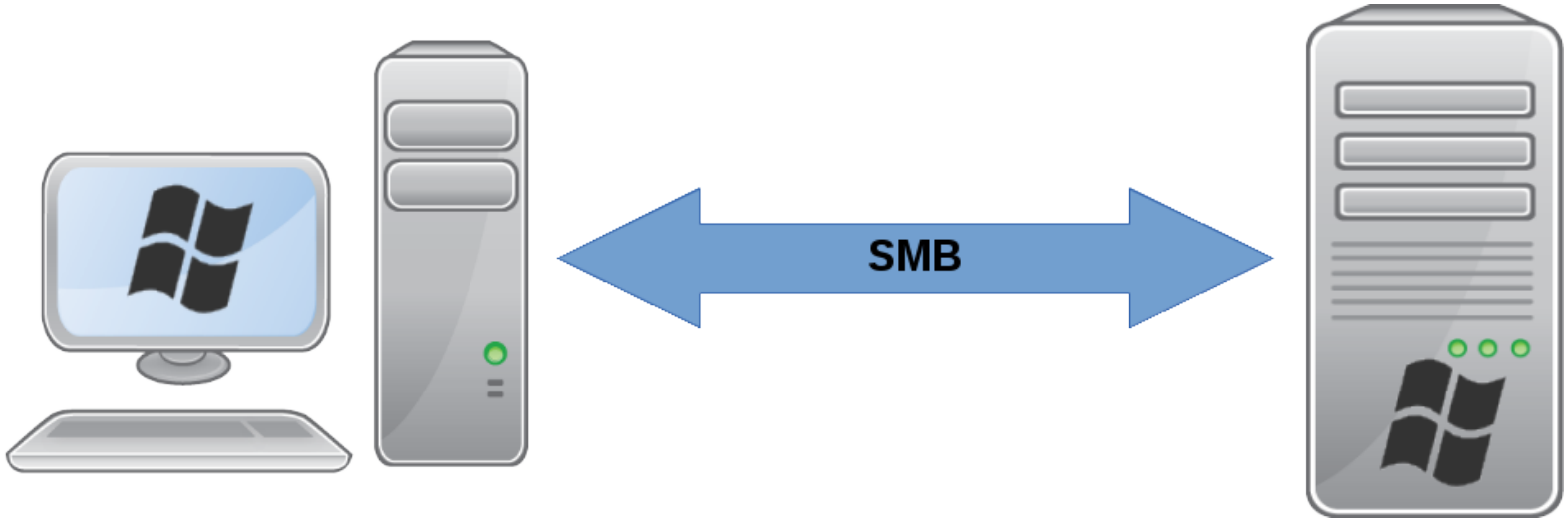
# The best CTDB bugs ever!

Amitay Isaacs <amitay@samba.org>
Martin Schwenke <martin@meltin.net>

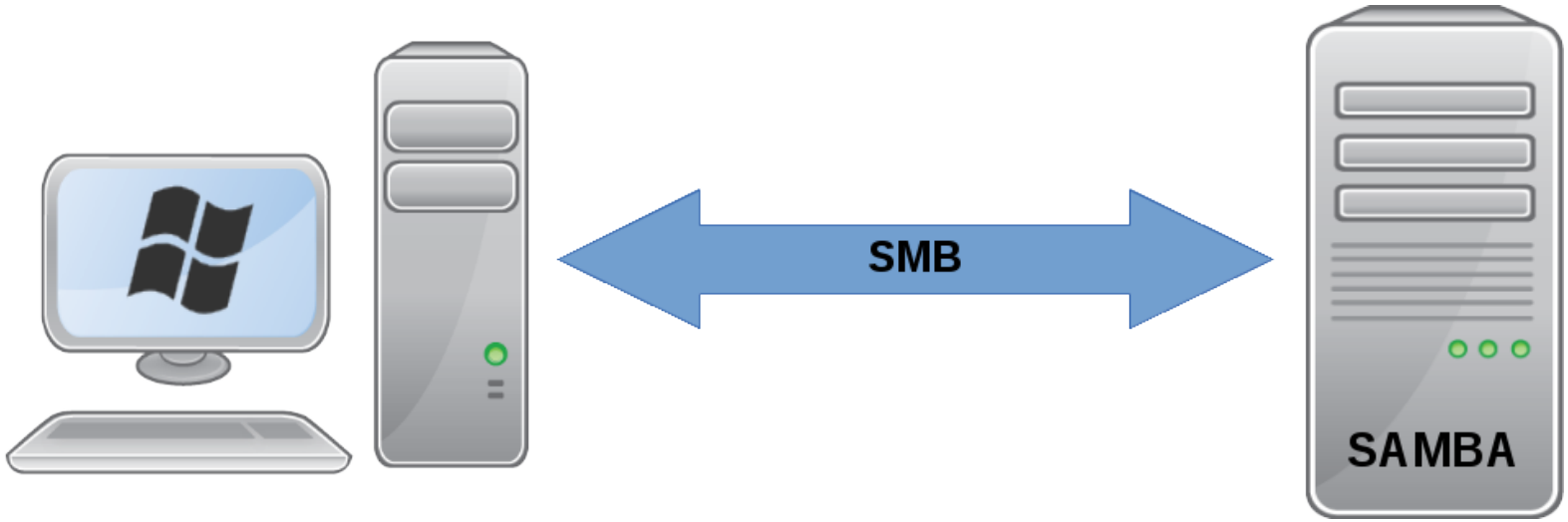
Samba Team
IBM (Australia Development Laboratory, Linux Technology Center)

# Samba



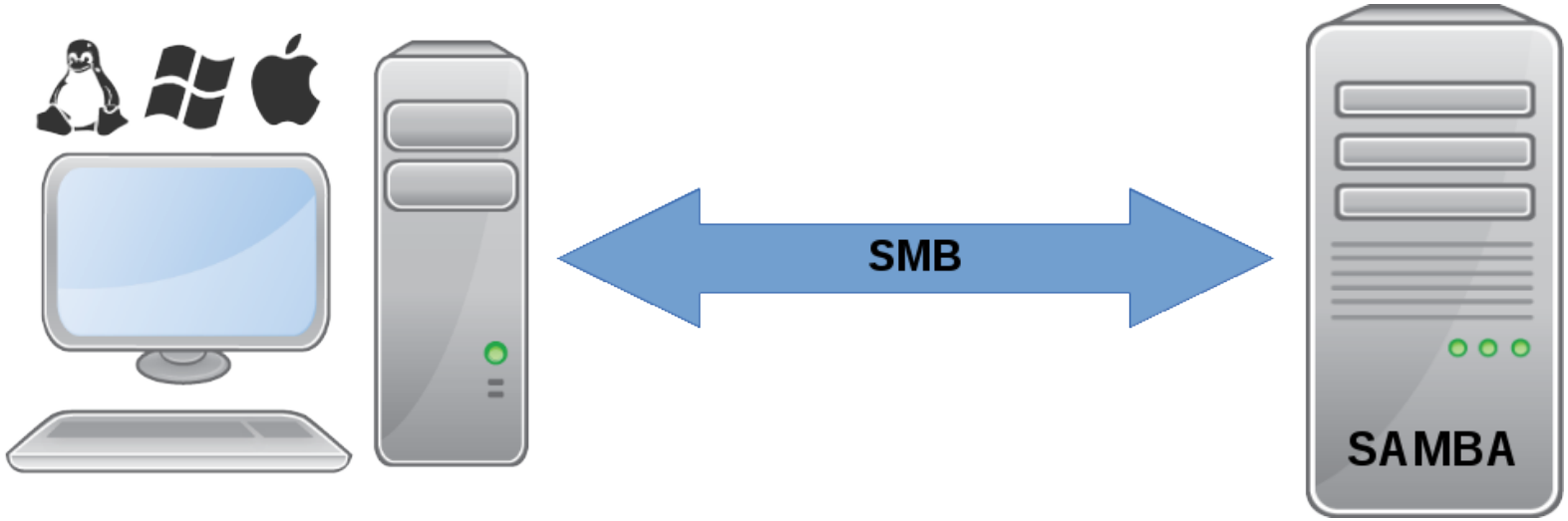

- Windows uses **SMB** protocol for file sharing

# Samba



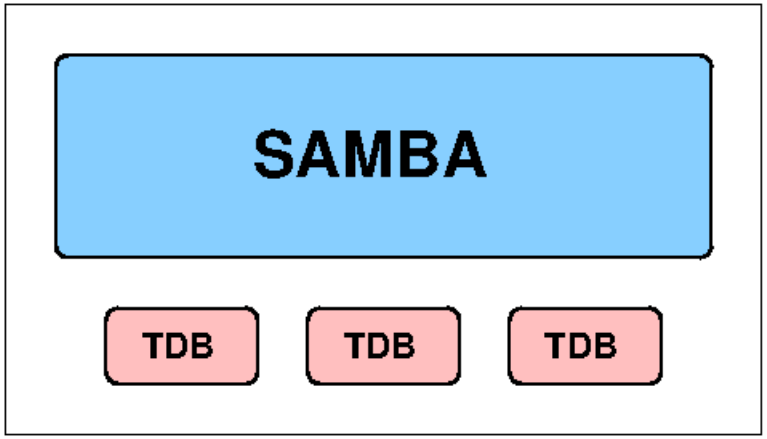

- Windows uses **SMB** protocol for file sharing
- Samba implements **SMB** protocol to support file sharing

# Samba



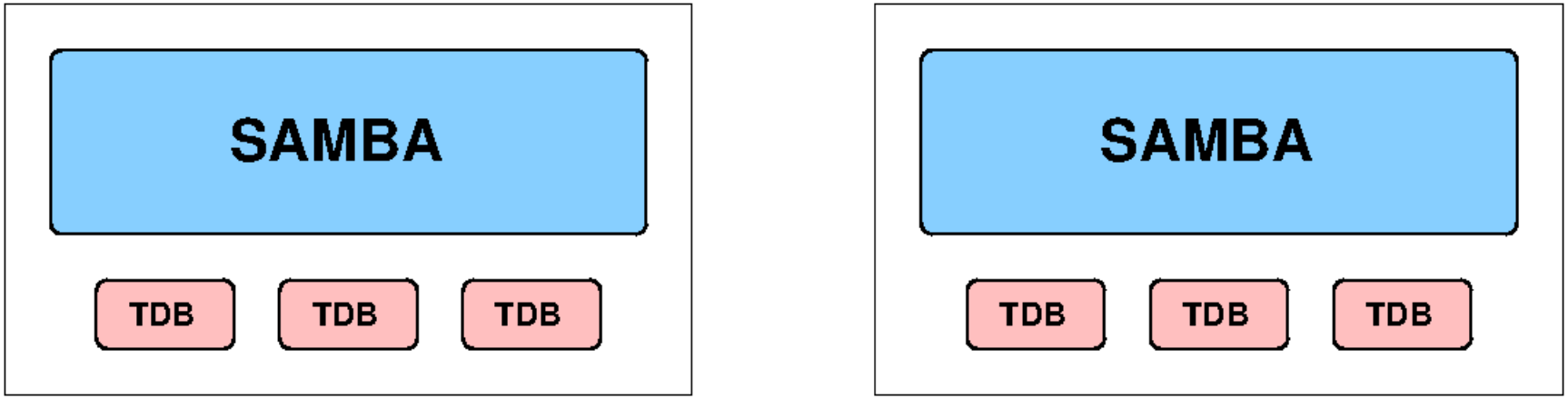

- Windows uses **SMB** protocol for file sharing
- Samba implements **SMB** protocol to support file sharing
- Multiple operating systems implement **SMB** client

# Clustering Samba



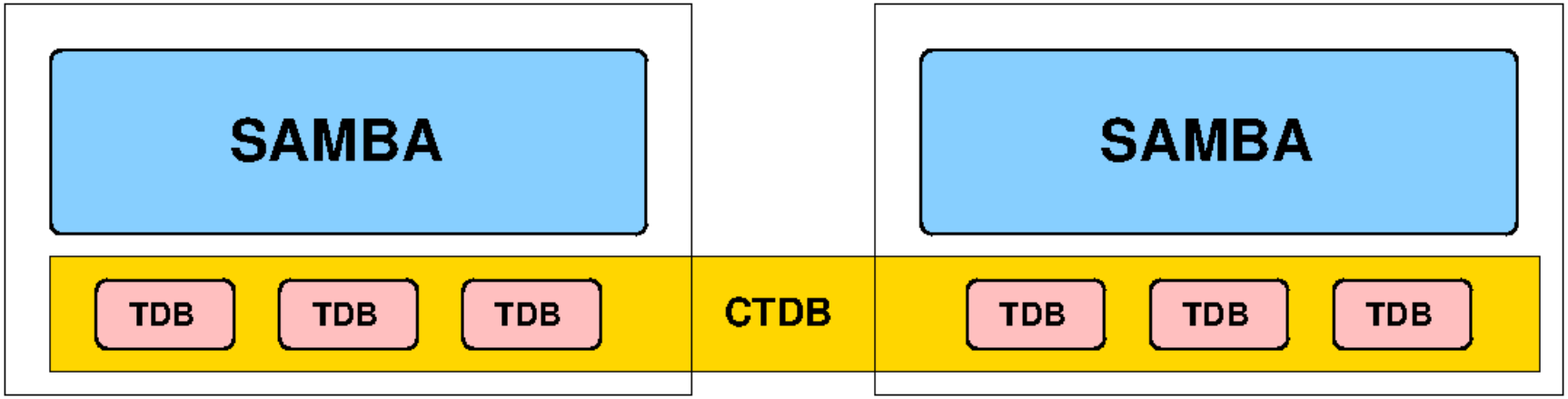

- Samba uses TDB databases to store meta data
  - e.g. active sessions, open files, locks

# Clustering Samba




- Samba uses TDB databases to store meta data
  - e.g. active sessions, open files, locks
- Each Samba server has a local copy of TDB databases

# Clustering Samba




- Samba uses TDB databases to store meta data
  - e.g. active sessions, open files, locks
- Each Samba server has a local copy of TDB databases
- CTDB manages TDB databases across multiple servers
  - Provides a **single database** view

# Building blocks of CTDB

1. **talloc**
   - hierarchical memory allocator
   - implements destructors
2. **tevent**
   - event handling system
   - based on poll/epoll
   - file descriptor events
   - timer events
3. **tdb**
   - file based database for key/value pairs
   - fcntl locks (read/write/transaction)
   - allow concurrent access from multiple processes

# CTDB

- Design
  - High performance
  - Main deamon is completely non-blocking

# CTDB

- Design
  - High performance
  - Main deamon is completely non-blocking
- Features
  - High Availability — active-active configuration
  - Load Balancing — round-robin DNS
  - IP allocation and failover
  - Services monitoring

# CTDB

- Design
  - High performance
  - Main deamon is completely non-blocking
- Features
  - High Availability — active-active configuration
  - Load Balancing — round-robin DNS
  - IP allocation and failover
  - Services monitoring
- Defects
  - Performance improvements
  - Integration testing

# Performance: Slow CTDB Response

**Problem**

When running clustered Samba under load,
CTDB daemon response becomes slower.

# Performance: Slow CTDB Response

## Evidence

```
# netstat -nt | grep -E "Proto|4379"
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.31.136.2:43341      172.31.136.3:4379       ESTABLISHED
tcp    16216      0 172.31.136.2:4379       172.31.132.2:57719      ESTABLISHED
tcp    43976      0 172.31.136.2:4379       172.31.136.3:43862      ESTABLISHED
tcp   410256      0 172.31.136.2:4379       172.31.132.3:42850      ESTABLISHED
tcp        0      0 172.31.136.2:43086      172.31.132.1:4379       ESTABLISHED
tcp        0    160 172.31.136.2:52568      172.31.132.3:4379       ESTABLISHED
tcp        0      0 172.31.136.2:48758      172.31.132.2:4379       ESTABLISHED
tcp    48464      0 172.31.136.2:4379       172.31.132.1:59573      ESTABLISHED
```

# Performance: Slow CTDB Response

## Evidence

```
# netstat -nt | grep -E "Proto|4379"
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.31.136.2:43341      172.31.136.3:4379       ESTABLISHED
tcp    16216      0 172.31.136.2:4379       172.31.132.2:57719      ESTABLISHED
tcp    43976      0 172.31.136.2:4379       172.31.136.3:43862      ESTABLISHED
tcp   410256      0 172.31.136.2:4379       172.31.132.3:42850      ESTABLISHED
tcp        0      0 172.31.136.2:43086      172.31.132.1:4379       ESTABLISHED
tcp        0    160 172.31.136.2:52568      172.31.132.3:4379       ESTABLISHED
tcp        0      0 172.31.136.2:48758      172.31.132.2:4379       ESTABLISHED
tcp    48464      0 172.31.136.2:4379       172.31.132.1:59573      ESTABLISHED
```

# Performance: Slow CTDB Response

## Evidence

```
# netstat -nt | grep -E "Proto|4379"
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.31.136.2:43341      172.31.136.3:4379       ESTABLISHED
tcp    16216      0 172.31.136.2:4379       172.31.132.2:57719      ESTABLISHED
tcp    43976      0 172.31.136.2:4379       172.31.136.3:43862      ESTABLISHED
tcp   410256      0 172.31.136.2:4379       172.31.132.3:42850      ESTABLISHED
tcp        0      0 172.31.136.2:43086      172.31.132.1:4379       ESTABLISHED
tcp        0    160 172.31.136.2:52568      172.31.132.3:4379       ESTABLISHED
tcp        0      0 172.31.136.2:48758      172.31.132.2:4379       ESTABLISHED
tcp    48464      0 172.31.136.2:4379       172.31.132.1:59573      ESTABLISHED
```

### Analysis

- How does CTDB read from sockets?

# Performance: Slow CTDB Response

## Evidence

```
# netstat -nt | grep -E "Proto|4379"
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.31.136.2:43341      172.31.136.3:4379       ESTABLISHED
tcp    16216      0 172.31.136.2:4379       172.31.132.2:57719      ESTABLISHED
tcp    43976      0 172.31.136.2:4379       172.31.136.3:43862      ESTABLISHED
tcp   410256      0 172.31.136.2:4379       172.31.132.3:42850      ESTABLISHED
tcp        0      0 172.31.136.2:43086      172.31.132.1:4379       ESTABLISHED
tcp        0    160 172.31.136.2:52568      172.31.132.3:4379       ESTABLISHED
tcp        0      0 172.31.136.2:48758      172.31.132.2:4379       ESTABLISHED
tcp    48464      0 172.31.136.2:4379       172.31.132.1:59573      ESTABLISHED
```

### Analysis

- How does CTDB read from sockets?
- Using a function for read events on sockets via tevent

# Performance: Slow CTDB Response

## queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    sz_bytes_req = sizeof(pkt_size);
    to_read = MIN(sz_bytes_req, num_ready);
    nread = read(queue->fd, data + queue->partial.length, to_read);

    pkt_size = *(uint32_t *)data;
    pkt_bytes_remaining = pkt_size - queue->partial.length;
    to_read = MIN(pkt_bytes, data + queue->partial.length, to_read);
    nread = read(queue->fd, data + queue->partial.length, to_read);
    queue->partial.length += nread;

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    sz_bytes_req = sizeof(pkt_size);
    to_read = MIN(sz_bytes_req, num_ready);
    nread = read(queue->fd, data + queue->partial.length, to_read);

    pkt_size = *(uint32_t *)data;
    pkt_bytes_remaining = pkt_size - queue->partial.length;
    to_read = MIN(pkt_bytes, data + queue->partial.length, to_read);
    nread = read(queue->fd, data + queue->partial.length, to_read);
    queue->partial.length += nread;

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    sz_bytes_req = sizeof(pkt_size);
    to_read = MIN(sz_bytes_req, num_ready);
    nread = read(queue->fd, data + queue->partial.length, to_read);

    pkt_size = *(uint32_t *)data;
    pkt_bytes_remaining = pkt_size - queue->partial.length;
    to_read = MIN(pkt_bytes, data + queue->partial.length, to_read);
    nread = read(queue->fd, data + queue->partial.length, to_read);
    queue->partial.length += nread;

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    sz_bytes_req = sizeof(pkt_size);
    to_read = MIN(sz_bytes_req, num_ready);
    nread = read(queue->fd, data + queue->partial.length, to_read);

    pkt_size = *(uint32_t *)data;
    pkt_bytes_remaining = pkt_size - queue->partial.length;
    to_read = MIN(pkt_bytes, data + queue->partial.length, to_read);
    nread = read(queue->fd, data + queue->partial.length, to_read);
    queue->partial.length += nread;

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    sz_bytes_req = sizeof(pkt_size);
    to_read = MIN(sz_bytes_req, num_ready);
    nread = read(queue->fd, data + queue->partial.length, to_read);

    pkt_size = *(uint32_t *)data;
    pkt_bytes_remaining = pkt_size - queue->partial.length;
    to_read = MIN(pkt_bytes, data + queue->partial.length, to_read);
    nread = read(queue->fd, data + queue->partial.length, to_read);
    queue->partial.length += nread;

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    sz_bytes_req = sizeof(pkt_size);
    to_read = MIN(sz_bytes_req, num_ready);
    nread = read(queue->fd, data + queue->partial.length, to_read);

    pkt_size = *(uint32_t *)data;
    pkt_bytes_remaining = pkt_size - queue->partial.length;
    to_read = MIN(pkt_bytes, data + queue->partial.length, to_read);
    nread = read(queue->fd, data + queue->partial.length, to_read);
    queue->partial.length += nread;

    queue->callback(data, pkt_size, queue->private_data);
}
```

**Analysis**

- Two **read** system calls to read a single packet

# Performance: Slow CTDB Response

## Improved queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    if (queue->buffer.length + num_ready > queue->buffer.size) { ... }

    nread = read(queue->fd, queue->buffer.data + queue->buffer.length,
                 num_ready);
    queue->buffer.length += nread;

    queue_process(queue);
}
```

# Performance: Slow CTDB Response

## Improved queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    if (queue->buffer.length + num_ready > queue->buffer.size) { ... }

    nread = read(queue->fd, queue->buffer.data + queue->buffer.length,
                 num_ready);
    queue->buffer.length += nread;

    queue_process(queue);
}
```

# Performance: Slow CTDB Response

## Improved queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    if (queue->buffer.length + num_ready > queue->buffer.size) { ... }

    nread = read(queue->fd, queue->buffer.data + queue->buffer.length,
                 num_ready);
    queue->buffer.length += nread;

    queue_process(queue);
}
```

**Analysis**

- Read all available data from socket

# Performance: Slow CTDB Response

## Improved queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    if (queue->buffer.length + num_ready > queue->buffer.size) { ... }

    nread = read(queue->fd, queue->buffer.data + queue->buffer.length,
                 num_ready);
    queue->buffer.length += nread;

    queue_process(queue);
}
```

**Analysis**

- Read all available data from socket

# Performance: Slow CTDB Response

## Improved queue_io_read code:

```
static void queue_io_read(struct ctdb_queue *queue)
{
    if (ioctl(queue->fd, FIONREAD, &num_ready) != 0) { ... }

    if (queue->buffer.length + num_ready > queue->buffer.size) { ... }

    nread = read(queue->fd, queue->buffer.data + queue->buffer.length,
                 num_ready);
    queue->buffer.length += nread;

    queue_process(queue);
}
```

**Analysis**

- Read all available data from socket
- Multiple packets can be read with a single **read** system call

# Performance: Slow CTDB Response

## queue_process code:

```
static void queue_process(struct ctdb_queue *queue)
{
    pkt_size = *(uint32_t *)queue->buffer.data;

    memcpy(data, queue->buffer.data, pkt_size);
    queue->buffer.length -= pkt_size;

    if (queue->buffer.length > 0) {
        tevent_schedule_immediate(queue->im, queue->ctdb->ev,
                                  queue_process, queue);
    }

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_process code:

```
static void queue_process(struct ctdb_queue *queue)
{
    pkt_size = *(uint32_t *)queue->buffer.data;

    memcpy(data, queue->buffer.data, pkt_size);
    queue->buffer.length -= pkt_size;

    if (queue->buffer.length > 0) {
        tevent_schedule_immediate(queue->im, queue->ctdb->ev,
                                  queue_process, queue);
    }

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

queue_process code:

```
static void queue_process(struct ctdb_queue *queue)
{
    pkt_size = *(uint32_t *)queue->buffer.data;

    memcpy(data, queue->buffer.data, pkt_size);
    queue->buffer.length -= pkt_size;

    if (queue->buffer.length > 0) {
        tevent_schedule_immediate(queue->im, queue->ctdb->ev,
                                  queue_process, queue);
    }

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_process code:

```
static void queue_process(struct ctdb_queue *queue)
{
    pkt_size = *(uint32_t *)queue->buffer.data;

    memcpy(data, queue->buffer.data, pkt_size);
    queue->buffer.length -= pkt_size;

    if (queue->buffer.length > 0) {
        tevent_schedule_immediate(queue->im, queue->ctdb->ev,
                                  queue_process, queue);
    }

    queue->callback(data, pkt_size, queue->private_data);
}
```

# Performance: Slow CTDB Response

## queue_process code:

```
static void queue_process(struct ctdb_queue *queue)
{
    pkt_size = *(uint32_t *)queue->buffer.data;

    memcpy(data, queue->buffer.data, pkt_size);
    queue->buffer.length -= pkt_size;

    if (queue->buffer.length > 0) {
        tevent_schedule_immediate(queue->im, queue->ctdb->ev,
                                  queue_process, queue);
    }

    queue->callback(data, pkt_size, queue->private_data);
}
```

### Analysis

- Multiple packets processed one-after-other

# Performance: Slow CTDB Response

## queue_process code:

```
static void queue_process(struct ctdb_queue *queue)
{
    pkt_size = *(uint32_t *)queue->buffer.data;

    memcpy(data, queue->buffer.data, pkt_size);
    queue->buffer.length -= pkt_size;

    if (queue->buffer.length > 0) {
        tevent_schedule_immediate(queue->im, queue->ctdb->ev,
                                  queue_process, queue);
    }

    queue->callback(data, pkt_size, queue->private_data);
}
```

### Analysis

- Multiple packets processed one-after-other
- What happens if there are large number of packets?

# Bug: `ctdb stop` fails

**Problem**

`ctdb stop` sometimes times out in our tests

# Bug: ctdb stop fails

## Evidence

```
# ctdb recmaster
3

# onnode 3 time ctdb stop
```

### Analysis

- Bug occurs when the "recovery master" node is stopped

# Bug: `ctdb stop` fails

## Evidence

```
# ctdb recmaster
3

# onnode 3 time ctdb stop
```

### Analysis

- Bug occurs when the "recovery master" node is stopped
- So run these commands repeatedly...

# Bug: ctdb stop fails

## Evidence

```
# ctdb recmaster
3

# onnode 3 time ctdb stop
Maximum runtime exceeded - exiting

real    2m0.001s
user    0m0.001s
sys     0m0.007s
```

### Analysis

- Bug occurs when the "recovery master" node is stopped
- So run these commands repeatedly...
- Eventually it times out...

# Bug: ctdb stop fails

## Evidence

```
# ctdb recmaster
3

# onnode 3 time ctdb stop
Maximum runtime exceeded - exiting

real    2m0.001s
user    0m0.001s
sys     0m0.007s
```

### Analysis

- Bug occurs when the "recovery master" node is stopped
- So run these commands repeatedly...
- Eventually it times out...
- `ctdb` command has 2 minute timeout to catch endless loops
- Code assumes this to retry potentially transient failures

# Bug: ctdb stop fails

## ctdb stop code:

```
do {
    ret = ctdb_ctrl_stop_node(ctdb, TIMELIMIT(), options.pnn);
    if (ret != 0) {
        ERR("Unable to stop node %u, try again\n", options.pnn);
    }
    sleep(1);
} while (!node_is_stopped(options.pnn));

ret = control_ipreallocate(ctdb, argc, argv);
if (ret != 0) {
    ERR("IP Reallocate failed on node %u\n", options.pnn);
    return ret;
}
return 0;
```

# Bug: ctdb stop fails

## ctdb stop code:

```
do {
    ret = ctdb_ctrl_stop_node(ctdb, TIMELIMIT(), options.pnn);
    if (ret != 0) {
        ERR("Unable to stop node %u, try again\n", options.pnn);
    }
    sleep(1);
} while (!node_is_stopped(options.pnn));

ret = control_ipreallocate(ctdb, argc, argv);
if (ret != 0) {
    ERR("IP Reallocate failed on node %u\n", options.pnn);
    return ret;
}
return 0;
```

### Analysis

- First we exclude this loop by printing a debug message after it.

# Bug: ctdb stop fails

## ctdb stop code:

```
do {
    ret = ctdb_ctrl_stop_node(ctdb, TIMELIMIT(), options.pnn);
    if (ret != 0) {
        ERR("Unable to stop node %u, try again\n", options.pnn);
    }
    sleep(1);
} while (!node_is_stopped(options.pnn));
ERR("Got here!!!\n");
ret = control_ipreallocate(ctdb, argc, argv);
if (ret != 0) {
    ERR("IP Reallocate failed on node %u\n", options.pnn);
    return ret;
}
return 0;
```

### Analysis

- First we exclude this loop by printing a debug message after it.

# Bug: ctdb stop fails

## ctdb stop code:

```
do {
    ret = ctdb_ctrl_stop_node(ctdb, TIMELIMIT(), options.pnn);
    if (ret != 0) {
        ERR("Unable to stop node %u, try again\n", options.pnn);
    }
    sleep(1);
} while (!node_is_stopped(options.pnn));
ERR("Got here!!!\n");
ret = control_ipreallocate(ctdb, argc, argv);
if (ret != 0) {
    ERR("IP Reallocate failed on node %u\n", options.pnn);
    return ret;
}
return 0;
```

### Analysis

- First we exclude this loop by printing a debug message after it.
- So, what happens in the `ipreallocate` code?

# Bug: ctdb stop fails

## ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ctdb_getrecmaster(ctdb_connection, options.pnn, &recmaster);

    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, recmaster, CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

# Bug: ctdb stop fails

## ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ctdb_getrecmaster(ctdb_connection, options.pnn, &recmaster);

    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, recmaster, CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

### Analysis

- Handler sets `ipreallocate_finished` when reply received

# Bug: ctdb stop fails

## ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ctdb_getrecmaster(ctdb_connection, options.pnn, &recmaster);

    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, recmaster, CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

**Analysis**

- Handler sets `ipreallocate_finished` when reply received
- Ask ctdbd on current node "who is the recovery master?"

# Bug: ctdb stop fails

## ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ctdb_getrecmaster(ctdb_connection, options.pnn, &recmaster);

    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, recmaster, CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

**Analysis**

- Handler sets `ipreallocate_finished` when reply received
- Ask ctdbd on current node "who is the recovery master?"
- Ask recovery master to do an "ipreallocate"

# Bug: ctdb stop fails

## ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ctdb_getrecmaster(ctdb_connection, options.pnn, &recmaster);

    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, recmaster, CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

**Analysis**

- Any other (non-recmaster) node will ignore this request

# Bug: ctdb stop fails

## ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ctdb_getrecmaster(ctdb_connection, options.pnn, &recmaster);

    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, recmaster, CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

**Analysis**

- Any other (non-recmaster) node will ignore this request
- So requests sent to the wrong node get no reply

# Bug: ctdb stop fails

## ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ctdb_getrecmaster(ctdb_connection, options.pnn, &recmaster);

    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, recmaster, CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

**Analysis**

- Any other (non-recmaster) node will ignore this request
- So requests sent to the wrong node get no reply
- Why would a request go to the wrong node?

# Bug: `ctdb stop` fails

## So, which node is the recovery master?

```
# onnode all ctdb recmaster
>> NODE: 10.0.0.61 <<
1
>> NODE: 10.0.0.62 <<
1
>> NODE: 10.0.0.63 <<
1
>> NODE: 10.0.0.64 <<
0
```

**Analysis**

- Ouch!

# Bug: `ctdb stop` fails

## So, which node is the recovery master?

```
# onnode all ctdb recmaster
>> NODE: 10.0.0.61 <<
1
>> NODE: 10.0.0.62 <<
1
>> NODE: 10.0.0.63 <<
1
>> NODE: 10.0.0.64 <<
0
```

### Analysis

- Ouch!
- Node 3 has been stopped so is "inactive"

# Bug: `ctdb stop` fails

## So, which node is the recovery master?

```
# onnode all ctdb recmaster
>> NODE: 10.0.0.61 <<
1
>> NODE: 10.0.0.62 <<
1
>> NODE: 10.0.0.63 <<
1
>> NODE: 10.0.0.64 <<
0
```

### Analysis

- Ouch!
- Node 3 has been stopped so is "inactive"
- Inactive nodes can't always identify the recovery master...

# Bug: ctdb stop fails

## Fixed ipreallocate code:

```
    ctdb_client_set_message_handler(..., ip_reallocate_handler, NULL);

again:
    ipreallocate_finished = false;
    ctdb_client_send_message(ctdb, CTDB_BROADCAST_CONNECTED,
                             CTDB_SRVID_TAKEOVER_RUN, data);
    /* Wait a while for a response, and then... */
    if (!ipreallocate_finished) {
        goto again;
    }

    return 0;
```

**Analysis**

- Don't ask "who is the recovery master?"
- Just broadcast to all nodes and the recovery master will reply

# Bug: Time travel fail

**Problem**

After setting the system clock forward 10 years and back again, things don't work well

# Bug: Time travel fail

## Evidence

```
# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
41.httpd             Status:OK    Duration:0.022 Wed Dec  4 16:58:41 2012
49.winbind           Status:OK    Duration:0.017 Wed Dec  4 16:58:41 2012
50.samba             Status:OK    Duration:0.056 Wed Dec  4 16:58:41 2012
...
91.lvs               Status:OK    Duration:0.019 Wed Dec  4 16:58:41 2012

# date
Wed Dec  4 18:22:19 EST 2012
```

# Bug: Time travel fail

## Evidence

```
# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
41.httpd                 Status:OK    Duration:0.022 Wed Dec  4 16:58:41 2012
49.winbind               Status:OK    Duration:0.017 Wed Dec  4 16:58:41 2012
50.samba                 Status:OK    Duration:0.056 Wed Dec  4 16:58:41 2012
...
91.lvs                   Status:OK    Duration:0.019 Wed Dec  4 16:58:41 2012

# date
Wed Dec  4 18:22:19 EST 2012
```

### Analysis

- Monitor events are supposed to run every 15 seconds
- No monitor event for over an hour!

# Bug: Time travel fail

## Some of that monitor code

```
static void ctdb_check_health(...)
{
    ret = ctdb_event_script_callback(ctdb,
            ctdb->monitor->monitor_context,
            ctdb_health_callback,
            ctdb, false,
            CTDB_EVENT_MONITOR, "%s", "");
}

static void ctdb_health_callback(...)
{
    event_add_timed(ctdb->ev,
            ctdb->monitor->monitor_context,
            timeval_current_ofs(next_interval, 0),
            ctdb_check_health, ctdb);
}
```

# Bug: Time travel fail

## Some of that monitor code

```
static void ctdb_check_health(...)
{
    ret = ctdb_event_script_callback(ctdb,
            ctdb->monitor->monitor_context,
            ctdb_health_callback,
            ctdb, false,
            CTDB_EVENT_MONITOR, "%s", "");
}

static void ctdb_health_callback(...)
{
    event_add_timed(ctdb->ev,
            ctdb->monitor->monitor_context,
            timeval_current_ofs(next_interval, 0),
            ctdb_check_health, ctdb);
}
```

**Analysis**

- When a monitor event finishes, it schedules the next one

# Bug: Time travel fail

- Repeated events: each iteration of an event schedules the next iteration.

# Bug: Time travel fail

- Repeated events: each iteration of an event schedules the next iteration.
- CTDB uses the tevent library implement its event handling.

# Bug: Time travel fail

- Repeated events: each iteration of an event schedules the next iteration.
- CTDB uses the tevent library implement its event handling.
- Events are scheduled at *absolute* times.

# Bug: Time travel fail

- Repeated events: each iteration of an event schedules the next iteration.
- CTDB uses the tevent library implement its event handling.
- Events are scheduled at *absolute* times.

**Solution?**

Travel forward 10 years in time to when relative event scheduling support has been added to tevent ...

# Performance: Node crash

**Problem**

With 5000 active SMB connections, node crashes

# Performance: Node Crash

## Evidence

```
# crash .../vmcore .../2.6.32-131.28.1.el6.x86_64/vmlinux
crash> kmem -i
                 PAGES        TOTAL      PERCENTAGE
 TOTAL MEM  12321940        47 GB           ----
      FREE    124575     486.6 MB    1% of TOTAL MEM
      USED  12197365      46.5 GB   98% of TOTAL MEM
    SHARED   4451875        17 GB   36% of TOTAL MEM
   BUFFERS       131       524 KB    0% of TOTAL MEM
    CACHED      6316      24.7 MB    0% of TOTAL MEM
      SLAB    531935         2 GB    4% of TOTAL MEM

TOTAL SWAP   1221474       4.7 GB           ----
 SWAP USED   1221474       4.7 GB  100% of TOTAL SWAP
 SWAP FREE         0            0    0% of TOTAL SWAP
```

# Performance: Node Crash

## Evidence

```
# crash .../vmcore .../2.6.32-131.28.1.el6.x86_64/vmlinux
crash> ps | awk '{ print $9 }' | sort | uniq -c | sort -n | tail
     24 python
     24 [rds_fmr_flushd/]
     43 cimprovagt
     64 console-kit-dae
     96 [ext4-dio-unwrit]
    512 [nfsd]
    929 mmfsd
   1283 ctdbd
   3613 conn-main.out
   4025 smbd
```

# Performance: Node Crash

## Evidence

```
# crash .../vmcore .../2.6.32-131.28.1.el6.x86_64/vmlinux
crash> ps | awk '{ print $9 }' | sort | uniq -c | sort -n | tail
     24 python
     24 [rds_fmr_flushd/]
     43 cimprovagt
     64 console-kit-dae
     96 [ext4-dio-unwrit]
    512 [nfsd]
    929 mmfsd
   1283 ctdbd
   3613 conn-main.out
   4025 smbd
```

### Analysis

- Why are there so many CTDB processes?

# Performance: Node Crash

## Evidence

```
# crash .../vmcore .../2.6.32-131.28.1.el6.x86_64/vmlinux
crash> ps | awk '{ print $9 }' | sort | uniq -c | sort -n | tail
     24 python
     24 [rds_fmr_flushd/]
     43 cimprovagt
     64 console-kit-dae
     96 [ext4-dio-unwrit]
    512 [nfsd]
    929 mmfsd
   1283 ctdbd
   3613 conn-main.out
   4025 smbd
```

### Analysis

- Why are there so many CTDB processes?
- Blocking record locks – `ctdb_lockwait()`

# Performance: Node Crash

## ctdb_lockwait code:

```
struct lockwait_handle *ctdb_lockwait(..., TDB_DATA key, ...)
{
    if (ctdb_db->pending_requests > 200) {
        DLIST_ADD_END(ctdb_db->lockwait_overflow, result, NULL);
        return ...;
    }

    result->child = ctdb_fork(ctdb_db->ctdb);
    if (result->child == 0) {
        c = tdb_chainlock(ctdb_db->ltdb->tdb, key);
        write(result->fd[1], &c, 1);
        sleep(...);
    }

    result->fde = tevent_add_fd(ctdb_db->ctdb->ev, result->fd[0],
                                EVENT_FD_READ, ... );
}
```

# Performance: Node Crash

## ctdb_lockwait code:

```
struct lockwait_handle *ctdb_lockwait(..., TDB_DATA key, ...)
{
    if (ctdb_db->pending_requests > 200) {
        DLIST_ADD_END(ctdb_db->lockwait_overflow, result, NULL);
        return ...;
    }

    result->child = ctdb_fork(ctdb_db->ctdb);
    if (result->child == 0) {
        c = tdb_chainlock(ctdb_db->ltdb->tdb, key);
        write(result->fd[1], &c, 1);
        sleep(...);
    }

    result->fde = tevent_add_fd(ctdb_db->ctdb->ev, result->fd[0],
                                EVENT_FD_READ, ... );
}
```

# Performance: Node Crash

## ctdb_lockwait code:

```
struct lockwait_handle *ctdb_lockwait(..., TDB_DATA key, ...)
{
    if (ctdb_db->pending_requests > 200) {
        DLIST_ADD_END(ctdb_db->lockwait_overflow, result, NULL);
        return ...;
    }

    result->child = ctdb_fork(ctdb_db->ctdb);
    if (result->child == 0) {
        c = tdb_chainlock(ctdb_db->ltdb->tdb, key);
        write(result->fd[1], &c, 1);
        sleep(...);
    }

    result->fde = tevent_add_fd(ctdb_db->ctdb->ev, result->fd[0],
                                EVENT_FD_READ, ... );
}
```

# Performance: Node Crash

## ctdb_lockwait code:

```
struct lockwait_handle *ctdb_lockwait(..., TDB_DATA key, ...)
{
    if (ctdb_db->pending_requests > 200) {
        DLIST_ADD_END(ctdb_db->lockwait_overflow, result, NULL);
        return ...;
    }

    result->child = ctdb_fork(ctdb_db->ctdb);
    if (result->child == 0) {
        c = tdb_chainlock(ctdb_db->ltdb->tdb, key);
        write(result->fd[1], &c, 1);
        sleep(...);
    }

    result->fde = tevent_add_fd(ctdb_db->ctdb->ev, result->fd[0],
                                EVENT_FD_READ, ... );
}
```

# Performance: Node Crash

## ctdb_lockwait code:

```
struct lockwait_handle *ctdb_lockwait(..., TDB_DATA key, ...)
{
    if (ctdb_db->pending_requests > 200) {
        DLIST_ADD_END(ctdb_db->lockwait_overflow, result, NULL);
        return ...;
    }

    result->child = ctdb_fork(ctdb_db->ctdb);
    if (result->child == 0) {
        c = tdb_chainlock(ctdb_db->ltdb->tdb, key);
        write(result->fd[1], &c, 1);
        sleep(...);
    }

    result->fde = tevent_add_fd(ctdb_db->ctdb->ev, result->fd[0],
                                EVENT_FD_READ, ... );
}
```

**Analysis**

- This explains number of processes, but not OOM!

# Performance: Node Crash

- Copy on write - **fork** does not copy memory immediately

# Performance: Node Crash

- Copy on write - **fork** does not copy memory immediately

**Clue**

```
# cat /proc/meminfo
...
PageTables:   16879739 kB
...
```

# Performance: Node Crash

- Copy on write - **fork** does not copy memory immediately
- But **fork** does copy page tables

**Clue**

```
# cat /proc/meminfo
...
PageTables:   16879739 kB
...
```

# Performance: Node Crash

- Copy on write - **fork** does not copy memory immediately
- But **fork** does copy page tables
- Solution – **vfork**

# Performance: Node Crash

- Copy on write - **fork** does not copy memory immediately
- But **fork** does copy page tables
- Solution – **vfork**

How does fork compare with vfork?

# Performance: Node Crash

- Copy on write - **fork** does not copy memory immediately
- But **fork** does copy page tables
- Solution – **vfork**

How does fork compare with vfork?

Time (in $\mu$s) required to create a child process

| Memory | 0M | 10M | 100M |
|---|---|---|---|
| **fork** | $41 \pm 3$ | $144 \pm 9$ | $997 \pm 14$ |
| **vfork** | $77 \pm 31$ | $81 \pm 37$ | $79 \pm 32$ |

# Performance: Node Crash

- Copy on write - **fork** does not copy memory immediately
- But **fork** does copy page tables
- Solution – **vfork**

How does fork compare with vfork?

Time (in $\mu$s) required to create a child process

| Memory | 0M | 10M | 100M |
|---|---|---|---|
| **fork** | $41 \pm 3$ | $144 \pm 9$ | $997 \pm 14$ |
| **vfork** | $77 \pm 31$ | $81 \pm 37$ | $79 \pm 32$ |

- Replace **fork** with **vfork**

# Performance: Node Crash

## Improved ctdb_lockwait code:

```
struct lockwait_handle *ctdb_lockwait(..., TDB_DATA key, ...)
{
    if (ctdb_db->pending_requests > 200) {
        DLIST_ADD_END(ctdb_db->lockwait_overflow, result, NULL);
        return ...;
    }

    result->child = vfork();
    if (result->child == 0) {
        execl("ctdb_lockwait_helper",  ... );
        _exit(1);
    }

    result->fde = tevent_add_fd(ctdb_db->ctdb->ev, result->fd[0],
                                EVENT_FD_READ, ... );
}
```

# Performance: Node Crash

## Improved ctdb_lockwait code:

```
struct lockwait_handle *ctdb_lockwait(..., TDB_DATA key, ...)
{
    if (ctdb_db->pending_requests > 200) {
        DLIST_ADD_END(ctdb_db->lockwait_overflow, result, NULL);
        return ...;
    }

    result->child = vfork();
    if (result->child == 0) {
        execl("ctdb_lockwait_helper",  ... );
        _exit(1);
    }

    result->fde = tevent_add_fd(ctdb_db->ctdb->ev, result->fd[0],
                                EVENT_FD_READ, ... );
}
```

**Analysis**

- Use light-weight helper process to obtain a record lock

# Performance: High CPU Consumption

**Problem**

With 5000 active SMB connections,
CTDB is consuming close to 100% CPU

# Performance: High CPU Consumption

## Evidence

```
# perf record -a -g -- sleep 60
```

# Performance: High CPU Consumption

## Evidence

```
# perf record -a -g -- sleep 60
# perf report
    15.43%        ctdbd  ctdbd                          [.] daemon_check_srvids
                  |
                  --- daemon_check_srvids
                      |
                      |--100.00%-- ctdb_control_dispatch
                      |          ctdb_request_control
                      |          ctdb_input_pkt
                      |          queue_next_trigger
                      |          tevent_common_loop_timer_delay
                      |          std_event_loop_once
                      |          _tevent_loop_once
                      |          tevent_common_loop_wait
                      |          _tevent_loop_wait
                      |          ctdb_start_daemon
                      |          main
                      |          __libc_start_main
```

# Performance: High CPU Consumption

## Evidence

```
# perf record -a -g -- sleep 60
# perf report
    15.43%        ctdbd  ctdbd                          [.] daemon_check_srvids
                  |
                  --- daemon_check_srvids
                      |
                      |--100.00%-- ctdb_control_dispatch
                      |          ctdb_request_control
                      |          ctdb_input_pkt
                      |          queue_next_trigger
                      |          tevent_common_loop_timer_delay
                      |          std_event_loop_once
                      |          _tevent_loop_once
                      |          tevent_common_loop_wait
                      |          _tevent_loop_wait
                      |          ctdb_start_daemon
                      |          main
                      |          __libc_start_main
```

**Analysis**

- CTDB is spending too much time in **daemon_check_srvids**

# Performance: High CPU Consumption

## daemon_check_srvids code:

```
int daemon_check_srvids(struct ctdb_context *ctdb, ... )
{

    for (i=0; i<num_ids; i++) {
         struct ctdb_message_list *ml;
         for (ml=ctdb->message_list; ml; ml=ml->next) {
             if (ml->srvid == ids[i]) {
                 break;
             }
         }
    }

}
```

# Performance: High CPU Consumption

## daemon_check_srvids code:

```
int daemon_check_srvids(struct ctdb_context *ctdb, ... )
{

    for (i=0; i<num_ids; i++) {
        struct ctdb_message_list *ml;
        for (ml=ctdb->message_list; ml; ml=ml->next) {
            if (ml->srvid == ids[i]) {
                break;
            }
        }
    }

}
```

**Analysis**

- Search given messages ids in a linked list

# Performance: High CPU Consumption

## daemon_check_srvids code:

```
int daemon_check_srvids(struct ctdb_context *ctdb, ... )
{

    for (i=0; i<num_ids; i++) {
         struct ctdb_message_list *ml;
         for (ml=ctdb->message_list; ml; ml=ml->next) {
             if (ml->srvid == ids[i]) {
                 break;
             }
         }
    }

}
```

**Analysis**

- Search given messages ids in a linked list
- Replace linked-list with a hash table

# Performance: High CPU Consumption

## Another bottleneck (from tevent library)

```
tevent_common_add_timer(..., struct timeval next_event, ...)
{

    te->next_event = next_event;

    for (cur_te = DLIST_TAIL(ev->timer_events);
         cur_te != NULL; cur_te = DLIST_PREV(cur_te)) {

        ret = tevent_timeval_compare(&te->next_event, &cur_te->next_event);
        if (ret < 0) continue;
        break;
    }

}
```

**Analysis**

- All the timer events are kept sorted in increasing time order

# Performance: High CPU Consumption

## Another bottleneck (from tevent library)

```
tevent_common_add_timer(..., struct timeval next_event, ...)
{

    te->next_event = next_event;

    for (cur_te = DLIST_TAIL(ev->timer_events);
         cur_te != NULL; cur_te = DLIST_PREV(cur_te)) {

        ret = tevent_timeval_compare(&te->next_event, &cur_te->next_event);
        if (ret < 0) continue;
        break;
    }

}
```

### Analysis

- All the timer events are kept sorted in increasing time order
- Mostly timer events are created for future, search from the end of the list

# Performance: High CPU Consumption

## Another bottleneck (from tevent library)

```
tevent_common_add_timer(..., struct timeval next_event, ...)
{

    te->next_event = next_event;

    for (cur_te = DLIST_TAIL(ev->timer_events);
         cur_te != NULL; cur_te = DLIST_PREV(cur_te)) {

        ret = tevent_timeval_compare(&te->next_event, &cur_te->next_event);
        if (ret < 0) continue;
        break;
    }

}
```

**Analysis**

- All the timer events are kept sorted in increasing time order
- Mostly timer events are created for future, search from the end of the list
- CTDB schedules lots of immediate events with zero time

# Performance: High CPU Consumption

## Another bottleneck (from tevent library)

```
tevent_common_add_timer(..., struct timeval next_event, ...)
{

    te->next_event = next_event;

    for (cur_te = DLIST_TAIL(ev->timer_events);
         cur_te != NULL; cur_te = DLIST_PREV(cur_te)) {

        ret = tevent_timeval_compare(&te->next_event, &cur_te->next_event);
        if (ret < 0) continue;
        break;
    }

}
```

**Analysis**

- All the timer events are kept sorted in increasing time order
- Mostly timer events are created for future, search from the end of the list
- CTDB schedules lots of immediate events with zero time
- Maintain a pointer to the last zero timeval entry in the list

# Bug: CTDB doesn't shut down

**Problem**

Sometimes `ctdb shutdown` does not shut down `ctdbd`

# Bug: CTDB doesn't shut down

## Evidence

```
# ctdb shutdown
# ctdb ping
response from 0 time=0.000076 sec  (2 clients)
```

# Bug: CTDB doesn't shut down

## Evidence

```
# ctdb shutdown
# ctdb ping
response from 0 time=0.000076 sec  (2 clients)
# sleep 30
```

# Bug: CTDB doesn't shut down

## Evidence

```
# ctdb shutdown
# ctdb ping
response from 0 time=0.000076 sec  (2 clients)
# sleep 30
# ctdb ping
response from 0 time=0.000055 sec  (2 clients)
```

# Bug: CTDB doesn't shut down

## Evidence

```
# ctdb shutdown
# ctdb ping
response from 0 time=0.000076 sec  (2 clients)
# sleep 30
# ctdb ping
response from 0 time=0.000055 sec  (2 clients)
```

**Analysis**

- The logs show that CTDB has started to shut down...

# Bug: CTDB doesn't shut down

## Evidence

```
# ctdb shutdown
# ctdb ping
response from 0 time=0.000076 sec  (2 clients)
# sleep 30
# ctdb ping
response from 0 time=0.000055 sec  (2 clients)
```

### Analysis

- The logs show that CTDB has started to shut down...
- ...but it never finishes

# Bug: CTDB doesn't shut down

## Shutdown sequence code

```
DEBUG(DEBUG_NOTICE,("Shutdown sequence commencing.\n"));
ctdb_set_runstate(ctdb, CTDB_RUNSTATE_SHUTDOWN);
ctdb_stop_recoverd(ctdb);
ctdb_stop_keepalive(ctdb);
ctdb_stop_monitoring(ctdb);
ctdb_release_all_ips(ctdb);
ctdb_event_script(ctdb, CTDB_EVENT_SHUTDOWN);
if (ctdb->methods != NULL) {
    ctdb->methods->shutdown(ctdb);
}
```

# Bug: CTDB doesn't shut down

## Shutdown sequence code

```
DEBUG(DEBUG_NOTICE,("Shutdown sequence commencing.\n"));
ctdb_set_runstate(ctdb, CTDB_RUNSTATE_SHUTDOWN);
ctdb_stop_recoverd(ctdb);
ctdb_stop_keepalive(ctdb);
ctdb_stop_monitoring(ctdb);
ctdb_release_all_ips(ctdb);
ctdb_event_script(ctdb, CTDB_EVENT_SHUTDOWN);
if (ctdb->methods != NULL) {
    ctdb->methods->shutdown(ctdb);
}
```

### Analysis

- First monitoring is stopped

# Bug: CTDB doesn't shut down

## Shutdown sequence code

```
DEBUG(DEBUG_NOTICE,("Shutdown sequence commencing.\n"));
ctdb_set_runstate(ctdb, CTDB_RUNSTATE_SHUTDOWN);
ctdb_stop_recoverd(ctdb);
ctdb_stop_keepalive(ctdb);
ctdb_stop_monitoring(ctdb);
ctdb_release_all_ips(ctdb);
ctdb_event_script(ctdb, CTDB_EVENT_SHUTDOWN);
if (ctdb->methods != NULL) {
    ctdb->methods->shutdown(ctdb);
}
```

### Analysis

- First monitoring is stopped
- Run `shutdown` event scripts

# Bug: CTDB doesn't shut down

## Shutdown sequence code

```
DEBUG(DEBUG_NOTICE,("Shutdown sequence commencing.\n"));
ctdb_set_runstate(ctdb, CTDB_RUNSTATE_SHUTDOWN);
ctdb_stop_recoverd(ctdb);
ctdb_stop_keepalive(ctdb);
ctdb_stop_monitoring(ctdb);
ctdb_release_all_ips(ctdb);
ctdb_event_script(ctdb, CTDB_EVENT_SHUTDOWN);
if (ctdb->methods != NULL) {
    ctdb->methods->shutdown(ctdb);
}
```

### Analysis

- First monitoring is stopped
- Run `shutdown` event scripts
- We get stuck in here — this never returns

# Bug: CTDB doesn't shut down

## Shutdown sequence code

```
DEBUG(DEBUG_NOTICE,("Shutdown sequence commencing.\n"));
ctdb_set_runstate(ctdb, CTDB_RUNSTATE_SHUTDOWN);
ctdb_stop_recoverd(ctdb);
ctdb_stop_keepalive(ctdb);
ctdb_stop_monitoring(ctdb);
ctdb_release_all_ips(ctdb);
ctdb_event_script(ctdb, CTDB_EVENT_SHUTDOWN);
if (ctdb->methods != NULL) {
    ctdb->methods->shutdown(ctdb);
}
```

### Analysis

- First monitoring is stopped
- Run `shutdown` event scripts
- We get stuck in here — this never returns
- So, how are the scripts run?

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- Scripts are launched asynchronously

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- Scripts are launched asynchronously
- The `done` flag is set to `false`

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- Scripts are launched asynchronously
- The `done` flag is set to `false`
- Spin and wait until the `done` flag is set

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- The callback is called on completion or timeout

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- The callback is called on completion or timeout
- The done flag is set

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- The callback is called on completion or timeout
- The done flag is set
- So why isn't `event_script_callback()` being called?

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- It is being called/freed before `ctdb_event_script_callback_v()` completes

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- It is being called/freed before `ctdb_event_script_callback_v()` completes
- Then the `done` flag is reset!

# Bug: CTDB doesn't shut down

```
int ctdb_event_script(...)
{
    ret = ctdb_event_script_callback_v(ctdb, ctdb,
            event_script_callback, &status, false, call, fmt, ap);

    status.done = false;
    while (status.done == false && event_loop_once(ctdb->ev) == 0) /* noop */;
}
static void event_script_callback(struct ctdb_context *ctdb, int status, void *
{
    struct callback_status *s = (struct callback_status *)private_data;
    s->done = true;
}
```

## Analysis

- It is being called/freed before `ctdb_event_script_callback_v()` completes
- Then the `done` flag is reset!
- Then this loop spins forever...

# Bug: CTDB doesn't shut down

```
static int ctdb_event_script_callback_v(...)
{
    state->callback = talloc(mem_ctx, struct event_script_callback);
    DLIST_ADD(ctdb->script_callbacks, state->callback);
    talloc_set_destructor(state->callback, remove_callback);

    /* Kill off any running monitor events to run this event. */
    if (ctdb->current_monitor) {
        talloc_free(ctdb->current_monitor);
        ctdb->current_monitor = NULL;
    }
}
```

# Bug: CTDB doesn't shut down

```
static int ctdb_event_script_callback_v(...)
{
    state->callback = talloc(mem_ctx, struct event_script_callback);
    DLIST_ADD(ctdb->script_callbacks, state->callback);
    talloc_set_destructor(state->callback, remove_callback);

    /* Kill off any running monitor events to run this event. */
    if (ctdb->current_monitor) {
        talloc_free(ctdb->current_monitor);
        ctdb->current_monitor = NULL;
    }
}
```

## Analysis

- Setup the state, including the callback

# Bug: CTDB doesn't shut down

```
static int ctdb_event_script_callback_v(...)
{
    state->callback = talloc(mem_ctx, struct event_script_callback);
    DLIST_ADD(ctdb->script_callbacks, state->callback);
    talloc_set_destructor(state->callback, remove_callback);

    /* Kill off any running monitor events to run this event. */
    if (ctdb->current_monitor) {
        talloc_free(ctdb->current_monitor);
        ctdb->current_monitor = NULL;
    }
}
```

## Analysis

- Setup the state, including the callback
- Cancel monitor event

# Bug: CTDB doesn't shut down

```
static int ctdb_event_script_callback_v(...)
{
    state->callback = talloc(mem_ctx, struct event_script_callback);
    DLIST_ADD(ctdb->script_callbacks, state->callback);
    talloc_set_destructor(state->callback, remove_callback);

    /* Kill off any running monitor events to run this event. */
    if (ctdb->current_monitor) {
        talloc_free(ctdb->current_monitor);
        ctdb->current_monitor = NULL;
    }
}
```

## Analysis

- Setup the state, including the callback
- Cancel monitor event
- Callback for `ctdb->current_monitor` has already been cancelled and freed when monitoring was stopped, but the pointer for the callback structure is left dangling... then...

# Bug: CTDB doesn't shut down

```
static int ctdb_event_script_callback_v(...)
{
    state->callback = talloc(mem_ctx, struct event_script_callback);
    DLIST_ADD(ctdb->script_callbacks, state->callback);
    talloc_set_destructor(state->callback, remove_callback);

    /* Kill off any running monitor events to run this event. */
    if (ctdb->current_monitor) {
        talloc_free(ctdb->current_monitor);
        ctdb->current_monitor = NULL;
    }
}
```

## Analysis

- New callback structure for the `shutdown` event gets allocated at the same location as the previous monitoring callback structure!!!

# Bug: CTDB doesn't shut down

```
static int ctdb_event_script_callback_v(...)
{
    state->callback = talloc(mem_ctx, struct event_script_callback);
    DLIST_ADD(ctdb->script_callbacks, state->callback);
    talloc_set_destructor(state->callback, remove_callback);

    /* Kill off any running monitor events to run this event. */
    if (ctdb->current_monitor) {
        talloc_free(ctdb->current_monitor);
        ctdb->current_monitor = NULL;
    }
}
```

## Analysis

- New callback structure for the `shutdown` event gets allocated at the same location as the previous monitoring callback structure!!!
- So it gets run and freed here...

# Bug: CTDB doesn't shut down

## Fixed code

```
static int ctdb_event_script_callback_v(...)
{
    /* Kill off any running monitor events to run this event. */
    if (ctdb->current_monitor) {
        talloc_free(ctdb->current_monitor);
        ctdb->current_monitor = NULL;
    }

    state->callback = talloc(mem_ctx, struct event_script_callback);
    DLIST_ADD(ctdb->script_callbacks, state->callback);
    talloc_set_destructor(state->callback, remove_callback);
}
```

**Analysis**

- Do things in a different order. . . :-)

# Bug: Missing routes

**Problem**

The NAT gateway feature won't work on some nodes. The routes just don't get created.

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
```

### Analysis

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

### Analysis

- An expected default route is missing

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

### Analysis

- An expected default route is missing
- The configuration correct...

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

### Analysis

- An expected default route is missing
- The configuration correct...
- Is the script that adds the routes being run?

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

### Analysis

- An expected default route is missing
- The configuration correct...
- Is the script that adds the routes being run?
- Yes!

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

### Analysis

- An expected default route is missing
- The configuration correct...
- Is the script that adds the routes being run?
- Yes!
- What happens if we trace it?

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

### Analysis

- An expected default route is missing
- The configuration correct...
- Is the script that adds the routes being run?
- Yes!
- What happens if we trace it?
- `ip route add 0.0.0.0/0 via 10.0.1.42 metric 10`

# Bug: Missing routes

## Evidence

```
# ip addr show dev eth0
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    inet 10.0.1.42/24 brd 10.0.1.255 scope global eth1
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

### Analysis

- An expected default route is missing
- The configuration correct...
- Is the script that adds the routes being run?
- Yes!
- What happens if we trace it?
- `ip route add 0.0.0.0/0 via 10.0.1.42 metric 10`
- That looks right...

# Bug: Missing routes

## Interactive debugging

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
/sbin/ip
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
/sbin/ip
/bin/ip
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
/sbin/ip
/bin/ip
# ls -l $(which -a ip)
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
/sbin/ip
/bin/ip
# ls -l $(which -a ip)
-rwxr-xr-x. 1 root root      0 Jul 11  2011 /bin/ip
-rwxr-xr-x. 1 root root 229776 Oct 18  2012 /sbin/ip
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
/sbin/ip
/bin/ip
# ls -l $(which -a ip)
-rwxr-xr-x. 1 root root      0 Jul 11  2011 /bin/ip
-rwxr-xr-x. 1 root root 229776 Oct 18  2012 /sbin/ip
# grep PATH /etc/ctdb/functions
PATH=/bin:/usr/bin:/usr/sbin:/sbin:$PATH
```

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
/sbin/ip
/bin/ip
# ls -l $(which -a ip)
-rwxr-xr-x. 1 root root      0 Jul 11  2011 /bin/ip
-rwxr-xr-x. 1 root root 229776 Oct 18  2012 /sbin/ip
# grep PATH /etc/ctdb/functions
PATH=/bin:/usr/bin:/usr/sbin:/sbin:$PATH
```

**Why is anything working?**

# Bug: Missing routes

## Interactive debugging

```
# ip route add 0.0.0.0/0 via 10.0.1.42 metric 10
# echo $?
0
# ip route show
10.0.1.0/24 dev eth1  proto kernel  scope link  src 10.0.1.42
default via 10.0.1.41 dev eth1 metric 10
# which ip
/sbin/ip
# which -a ip
/sbin/ip
/bin/ip
# ls -l $(which -a ip)
-rwxr-xr-x. 1 root root      0 Jul 11  2011 /bin/ip
-rwxr-xr-x. 1 root root 229776 Oct 18  2012 /sbin/ip
# grep PATH /etc/ctdb/functions
PATH=/bin:/usr/bin:/usr/sbin:/sbin:$PATH
```

**Why is anything working?**

- Absolute path `/sbin/ip` is used most of the time

# Bug: Missing routes

## Fixes

# Bug: Missing routes

## Fixes

- `rm /bin/ip`

# Bug: Missing routes

## Fixes

- `rm /bin/ip`
- Don't set `PATH` in `/etc/ctdb/functions`

# Bug: Rogue IP addresses

**Problem**

Log is full of repeated messages and monitoring hasn't run for days

# Bug: Rogue IP addresses

## Evidence

```
# tail /var/log/messages
ctdbd: Forced running of eventscripts with arguments ipreallocated
ctdbd: Monitoring event was cancelled
[Every second!]

# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
50.samba             Status:OK    Duration:0.056 Sat Jul  7 15:57:48 2012
...
91.lvs               Status:OK    Duration:0.019 Sat Jul  7 15:57:48 2012

# date
Tue Jul 10 12:21:45 EST 2012
```

# Bug: Rogue IP addresses

## Evidence

```
# tail /var/log/messages
ctdbd: Forced running of eventscripts with arguments ipreallocated
ctdbd: Monitoring event was cancelled
[Every second!]

# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
50.samba             Status:OK    Duration:0.056 Sat Jul  7 15:57:48 2012
...
91.lvs               Status:OK    Duration:0.019 Sat Jul  7 15:57:48 2012

# date
Tue Jul 10 12:21:45 EST 2012
```

# Bug: Rogue IP addresses

## Evidence

```
# tail /var/log/messages
ctdbd: Forced running of eventscripts with arguments ipreallocated
ctdbd: Monitoring event was cancelled
[Every second!]

# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
50.samba             Status:OK    Duration:0.056 Sat Jul  7 15:57:48 2012
...
91.lvs               Status:OK    Duration:0.019 Sat Jul  7 15:57:48 2012

# date
Tue Jul 10 12:21:45 EST 2012
```

# Bug: Rogue IP addresses

## Evidence

```
# tail /var/log/messages
ctdbd: Forced running of eventscripts with arguments ipreallocated
ctdbd: Monitoring event was cancelled
[Every second!]

# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
50.samba             Status:OK    Duration:0.056 Sat Jul  7 15:57:48 2012
...
91.lvs               Status:OK    Duration:0.019 Sat Jul  7 15:57:48 2012

# date
Tue Jul 10 12:21:45 EST 2012
```

### Analysis

- No sign of time travel!

# Bug: Rogue IP addresses

## Evidence

```
# tail /var/log/messages
ctdbd: Forced running of eventscripts with arguments ipreallocated
ctdbd: Monitoring event was cancelled
[Every second!]

# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
50.samba              Status:OK    Duration:0.056 Sat Jul  7 15:57:48 2012
...
91.lvs                Status:OK    Duration:0.019 Sat Jul  7 15:57:48 2012

# date
Tue Jul 10 12:21:45 EST 2012
```

### Analysis

- No sign of time travel!
- What is generating all those `ipreallocate` events?

# Bug: Rogue IP addresses

## Evidence

```
# tail /var/log/messages
ctdbd: Forced running of eventscripts with arguments ipreallocated
ctdbd: Monitoring event was cancelled
[Every second!]

# ctdb scriptstatus
19 scripts were executed last monitor cycle
...
50.samba             Status:OK    Duration:0.056 Sat Jul  7 15:57:48 2012
...
91.lvs               Status:OK    Duration:0.019 Sat Jul  7 15:57:48 2012

# date
Tue Jul 10 12:21:45 EST 2012
```

### Analysis

- No sign of time travel!
- What is generating all those `ipreallocate` events?
- Nothing on recovery master. . . check logs on all other nodes. . .

# Bug: Rogue IP addresses

## On one node we find this:

```
# tail /var/log/messages
ctdbd: recoverd:We are still serving a public address '192.168.123.45'
  that we should not be serving.
ctdbd: recoverd:Trigger takeoverrun
ctdbd: release_ip of IP 192.168.123.45 is known to the kernel, but we have no
  interface assigned, has someone manually configured it? Ignore for
  now.
[Every second]
```

# Bug: Rogue IP addresses

## On one node we find this:

```
# tail /var/log/messages
ctdbd: recoverd:We are still serving a public address '192.168.123.45'
  that we should not be serving.
ctdbd: recoverd:Trigger takeoverrun
ctdbd: release_ip of IP 192.168.123.45 is known to the kernel, but we have no
  interface assigned, has someone manually configured it? Ignore for
  now.
[Every second]
```

### Analysis

- Why doesn't CTDB know about the IP address?

# Bug: Rogue IP addresses

## On one node we find this:

```
# tail /var/log/messages
ctdbd: recoverd:We are still serving a public address '192.168.123.45'
  that we should not be serving.
ctdbd: recoverd:Trigger takeoverrun
ctdbd: release_ip of IP 192.168.123.45 is known to the kernel, but we have no
  interface assigned, has someone manually configured it? Ignore for
  now.
[Every second]
```

### Analysis

- Why doesn't CTDB know about the IP address?
- Misconfiguration?

# Bug: Rogue IP addresses

## On one node we find this:

```
# tail /var/log/messages
ctdbd: recoverd:We are still serving a public address '192.168.123.45'
  that we should not be serving.
ctdbd: recoverd:Trigger takeoverrun
ctdbd: release_ip of IP 192.168.123.45 is known to the kernel, but we have no
  interface assigned, has someone manually configured it? Ignore for
  now.
[Every second]
```

### Analysis

- Why doesn't CTDB know about the IP address?
- Misconfiguration?
- If CTDB recognises the problem then why doesn't it fix it?

# Bug: Rogue IP addresses

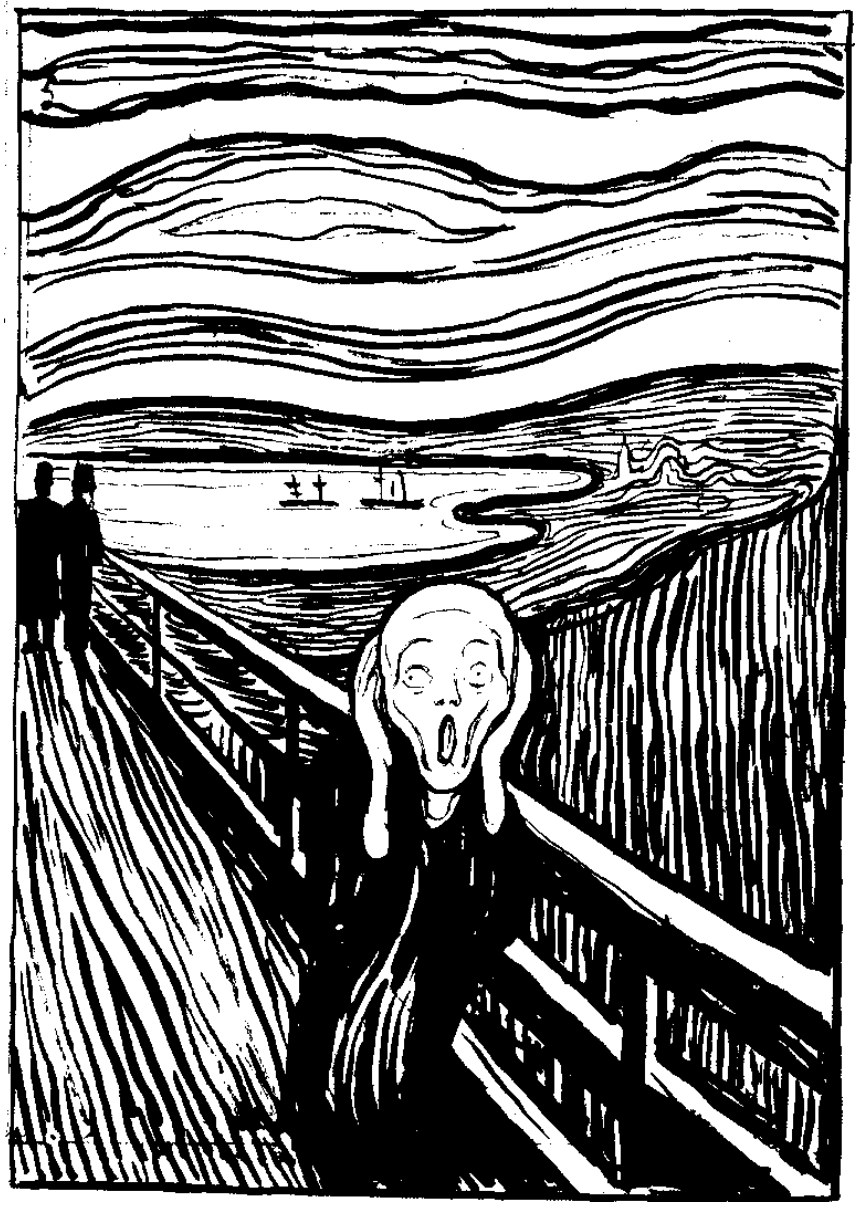
## On one node we find this:

```
# tail /var/log/messages
ctdbd: recoverd:We are still serving a public address '192.168.123.45'
  that we should not be serving.
ctdbd: recoverd:Trigger takeoverrun
ctdbd: release_ip of IP 192.168.123.45 is known to the kernel, but we have no
  interface assigned, has someone manually configured it? Ignore for
  now.
[Every second]
```

### Analysis

- Why doesn't CTDB know about the IP address?
- Misconfiguration?
- If CTDB recognises the problem then why doesn't it fix it?
- . . . instead of just going. . .

# Bug: Rogue IP addresses



# Bug: Rogue IP addresses

**Clue**

The logs showed a release IP shortly followed by a take IP, so let's look at the relevant code...

# Bug: Rogue IP addresses

**Clue**

The logs showed a release IP shortly followed by a take IP, so let's look at the relevant code...

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
```

# Bug: Rogue IP addresses

**Clue**

The logs showed a release IP shortly followed by a take IP, so let's look at the relevant code...

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);
```

# Bug: Rogue IP addresses

**Clue**

The logs showed a release IP shortly followed by a take IP, so let's look at the relevant code...

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

# Bug: Rogue IP addresses

**Clue**

The logs showed a release IP shortly followed by a take IP, so let's look at the relevant code...

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

# Bug: Rogue IP addresses

## Clue

The logs showed a release IP shortly followed by a take IP, so let's look at the relevant code...

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

## Analysis

- That seems pretty straightforward!

# Bug: Rogue IP addresses

**Question**

However, what happens if the order of events is slightly different?

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

# Bug: Rogue IP addresses

**Question**

However, what happens if the order of events is slightly different?

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

# Bug: Rogue IP addresses

**Question**

However, what happens if the order of events is slightly different?

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

# Bug: Rogue IP addresses

**Question**

However, what happens if the order of events is slightly different?

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

# Bug: Rogue IP addresses

**Question**

However, what happens if the order of events is slightly different?

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

# Bug: Rogue IP addresses

## Question

However, what happens if the order of events is slightly different?

```
ctdb_control_release_ip():
    ret = ctdb_event_script_callback(..., release_ip_callback, ...,
                                     CTDB_EVENT_RELEASE_IP, ...);
release_ip_callback():
    ctdb_vnn_unassign_iface(ctdb, state->vnn);

ctdb_control_takeover_ip():
    ret = ctdb_vnn_assign_iface(ctdb, vnn);
    ...
    ret = ctdb_event_script_callback(..., ctdb_do_takeip_callback, ...,
                                     CTDB_EVENT_TAKE_IP, ...);
```

## Analysis

- Aarrgghhh!!!

# Bug: Rogue IP addresses

**Workaround**

Ask OS for interface, remove IP address as requested

```
-       if (vnn->iface == NULL) {
-           DEBUG(DEBUG_ERR,(__location__ " release_ip of IP %s is known to the
-                "but we have no interface assigned, has someone manually confi
-                ctdb_addr_to_str(&vnn->public_address)));
+       iface = ctdb_sys_find_ifname(&pip->addr);
+       if (iface == NULL) {
+           DEBUG(DEBUG_ERR, ("Could not find which interface the ip address is
            return 0;
    }

    DEBUG(DEBUG_NOTICE,("Release of IP %s/%u on interface %s  node:%d\n",
        ctdb_addr_to_str(&pip->addr),
            vnn->public_netmask_bits,
-           ctdb_vnn_iface_string(vnn),
+           iface,
        pip->pnn));
```

# Bug: Rogue IP addresses

## Fixes

# Bug: Rogue IP addresses

## Fixes

- Stop controls for the same IP address from colliding in-flight. This is a big hammer, but it definitely avoids the problem!

# Bug: Rogue IP addresses

## Fixes

- Stop controls for the same IP address from colliding in-flight. This is a big hammer, but it definitely avoids the problem!
- Fail `release_ip_callback()` if the IP address is (still) present on an interface.

# Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?