CTDB: Where to from here and how can we get there?

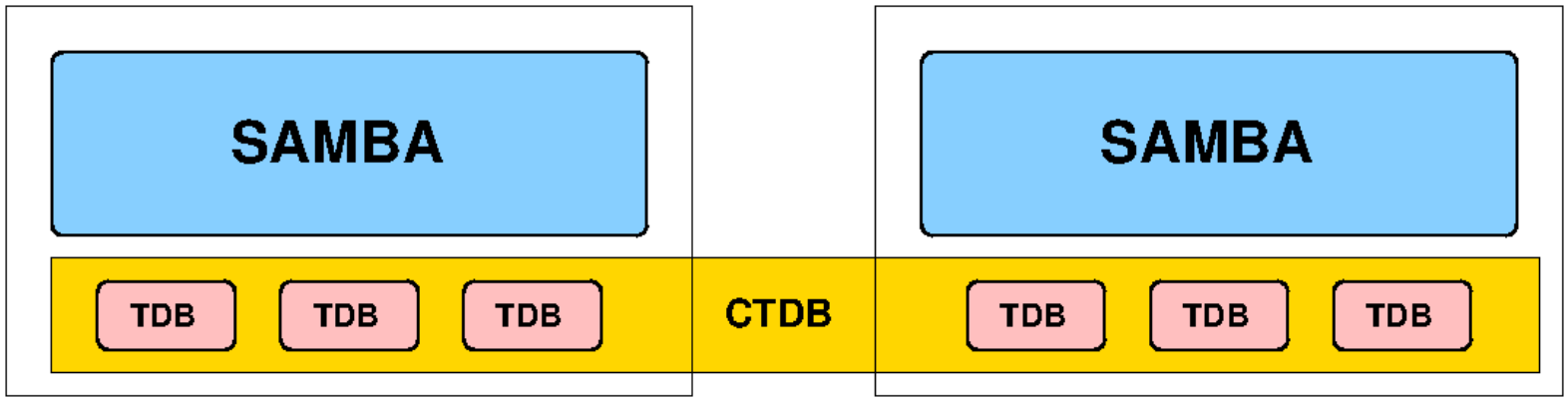
Martin Schwenke <martin@meltin.net>

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

What are we talking about?

What are we talking about?

What is CTDB?



What are we talking about?

What does CTDB do?

What are we talking about?

What does CTDB do?

- Cluster membership and leadership

What are we talking about?

What does CTDB do?

- Cluster membership and leadership
- Cluster database and database recovery

What are we talking about?

What does CTDB do?

- Cluster membership and leadership
- Cluster database and database recovery
- Cluster-wide messaging transport for Samba

What are we talking about?

What does CTDB do?

- Cluster membership and leadership
- Cluster database and database recovery
- Cluster-wide messaging transport for Samba
- Service management and monitoring

What are we talking about?

What does CTDB do?

- Cluster membership and leadership
- Cluster database and database recovery
- Cluster-wide messaging transport for Samba
- Service management and monitoring
- IP address management, failover and consistency checking

Where are we?

Where are we?

Current architecture

Where are we?

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

Where are we?

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

- Main daemon
- Recovery daemon

Where are we?

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

- Main daemon
- Recovery daemon

CTDB processes

Ephemeral processes to avoid blocking the main daemon

Where are we?

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

- Main daemon
- Recovery daemon

CTDB processes

Ephemeral processes to avoid blocking the main daemon

- Lock helper
- Event helper
- Vacuuming
- Persistent transaction
- Read-only record revocation
- State change notification
- Recovery lock sanity check
- Reloading public IP address configuration
- Database traverse

Where are we?

Mapping function to daemon

Where are we?

Mapping function to daemon

Main daemon

Recovery daemon

Where are we?

Mapping function to daemon

Main daemon

- **Cluster membership**

Recovery daemon

- **Cluster leadership**

Where are we?

Mapping function to daemon

Main daemon

- Cluster membership
- **Cluster database access**

Recovery daemon

- Cluster leadership
- **Cluster database recovery**

Where are we?

Mapping function to daemon

Main daemon

- Cluster membership
- Cluster database access
- **Cluster wide messaging transport**

Recovery daemon

- Cluster leadership
- Cluster database recovery

Where are we?

Mapping function to daemon

Main daemon

- Cluster membership
- Cluster database access
- Cluster wide messaging transport
- **Public IP address management**

Recovery daemon

- Cluster leadership
- Cluster database recovery
- **Public IP address failover and consistency checking**

Where are we?

Mapping function to daemon

Main daemon

- Cluster membership
- Cluster database access
- Cluster wide messaging transport
- Public IP address management
- **Service management**

Recovery daemon

- Cluster leadership
- Cluster database recovery
- Public IP address failover and consistency checking

Where are we?

Design Limitations

Where are we?

Design Limitations

- Main daemon and recovery daemon overloaded
  - Mix of time critical and non-critical in single daemon
  - Difficult to maintain in asynchronous, non-blocking design

Where are we?

Design Limitations

- Main daemon and recovery daemon overloaded
  - Mix of time critical and non-critical in single daemon
  - Difficult to maintain in asynchronous, non-blocking design
- Communication bottleneck
  - All messages must pass through (single threaded) main daemon

Where are we?

Design Limitations

- Main daemon and recovery daemon overloaded
  - Mix of time critical and non-critical in single daemon
  - Difficult to maintain in asynchronous, non-blocking design
- Communication bottleneck
  - All messages must pass through (single threaded) main daemon
- Cluster leader election
  - Each node tries to become leader on starting up

Where are we?

Design Limitations

- Main daemon and recovery daemon overloaded
  - Mix of time critical and non-critical in single daemon
  - Difficult to maintain in asynchronous, non-blocking design
- Communication bottleneck
  - All messages must pass through (single threaded) main daemon
- Cluster leader election
  - Each node tries to become leader on starting up
  - Does not scale with number of nodes!

Where are we?

Design Limitations

- Main daemon and recovery daemon overloaded
  - Mix of time critical and non-critical in single daemon
  - Difficult to maintain in asynchronous, non-blocking design
- Communication bottleneck
  - All messages must pass through (single threaded) main daemon
- Cluster leader election
  - Each node tries to become leader on starting up
  - Does not scale with number of nodes!
- Database recovery
  - Cluster leader recovers databases one at a time

Where are we?

Design Limitations...

- Centralised state
  - Some state is in main daemon but is used in recovery daemon

Where are we?

Design Limitations...

- Centralised state
  - Some state is in main daemon but is used in recovery daemon
- Tight coupling
  - Membership, service health, IP allocation are tightly coupled
  - Also consider cluster leader elections, database recovery, IP allocation, ...

Where are we?

Implementation Limitations

Where are we?

Implementation Limitations

- Protocol is “structs on the wire”
  - 32-bit vs 64-bit, not endian-neutral
  - Hand-marshalling of structures

Where are we?

Implementation Limitations

- Protocol is "structs on the wire"
  - 32-bit vs 64-bit, not endian-neutral
  - Hand-marshalling of structures
- Simpler protocol – single packet request/response
  - Streams / Large packets (e.g. multiple database records)
  - Large data buffer (talloc), Large send/recv (socket handling)

Where are we?

Implementation Limitations

- Protocol is "structs on the wire"
  - 32-bit vs 64-bit, not endian-neutral
  - Hand-marshalling of structures
- Simpler protocol – single packet request/response
  - Streams / Large packets (e.g. multiple database records)
  - Large data buffer (talloc), Large send/recv (socket handling)
- No (internal) messaging framework
  - Fire-and-forget method of communication with recovery daemon
  - First hurdle for multi-daemon design

Where are we?

Implementation Limitations

- Protocol is "structs on the wire"
  - 32-bit vs 64-bit, not endian-neutral
  - Hand-marshalling of structures
- Simpler protocol – single packet request/response
  - Streams / Large packets (e.g. multiple database records)
  - Large data buffer (talloc), Large send/recv (socket handling)
- No (internal) messaging framework
  - Fire-and-forget method of communication with recovery daemon
  - First hurdle for multi-daemon design
- Unstructured CLI and configuration

Where are we?

So?

Where are we?

So?

Big Problems

Where are we?

So?

Big Problems

- Limitations:
  - Scalability
  - Database recovery

Where are we?

So?

Big Problems

- Limitations:
  - Scalability
  - Database recovery
- Maintainability:
  - Organic growth: hacks and band-aids
  - *Ad hoc* data structures
  - *Ad hoc* data replication (connection info, NFS locks)

Where are we?

So?

Big Problems

- Limitations:
  - Scalability
  - Database recovery
- Maintainability:
  - Organic growth: hacks and band-aids
  - *Ad hoc* data structures
  - *Ad hoc* data replication (connection info, NFS locks)

Catch 22

Where are we?

So?

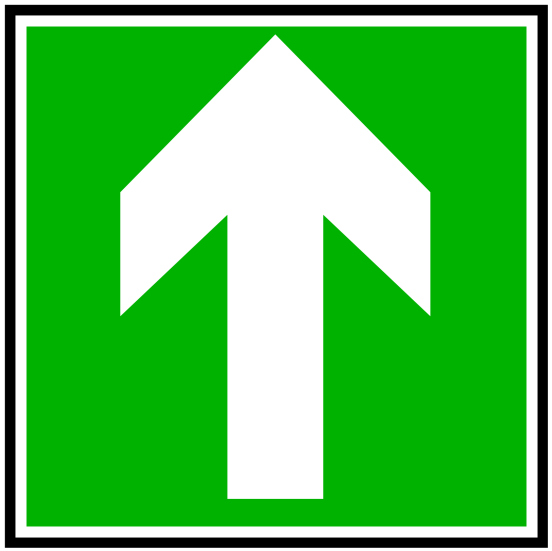
Big Problems

- Limitations:
  - Scalability
  - Database recovery
- Maintainability:
  - Organic growth: hacks and band-aids
  - *Ad hoc* data structures
  - *Ad hoc* data replication (connection info, NFS locks)

Catch 22

- We need help...
- However, barrier to entry is high!

Where to?



Where to?

Separate functionality in individual daemons

Where to?

Separate functionality in individual daemons

- Cluster management daemon

Where to?

Separate functionality in individual daemons

- Cluster management daemon
- Public IP address daemon

Where to?

Separate functionality in individual daemons

- Cluster management daemon
- Public IP address daemon
- Service management daemon

Where to?

Separate functionality in individual daemons

- Cluster management daemon
- Public IP address daemon
- Service management daemon
- Database daemon

Where to?

Separate functionality in individual daemons

- Cluster management daemon
- Public IP address daemon
- Service management daemon
- Database daemon
- ...

Where to?

Cluster management daemon

Where to?

Cluster management daemon

- Membership:
  - **Connected** according to heartbeat or similar
    - **Active** if not banned, administratively stopped

Where to?

Cluster management daemon

- Membership:
  - **Connected** according to heartbeat or similar
    - **Active** if not banned, administratively stopped
- Leadership:
  - Leader coordinates database recovery
  - Leader coordinates public IP address (re)allocation

Where to?

Cluster management daemon

- Membership:
  - **Connected** according to heartbeat or similar
    - **Active** if not banned, administratively stopped
- Leadership:
  - Leader coordinates database recovery
  - Leader coordinates public IP address (re)allocation
- Can we support etcd, Heartbeat (or similar) as an alternative?

Where to?

Public IP address daemon

Where to?

Public IP address daemon

- Single daemon with public IP address:
  - Management
  - Failover
  - Consistency checking

Where to?

Public IP address daemon

- Single daemon with public IP address:
  - Management
  - Failover
  - Consistency checking
- Simple management and status CLI

Where to?

Public IP address daemon

- Single daemon with public IP address:
  - Management
  - Failover
  - Consistency checking
- Simple management and status CLI
- Simple IP (re)allocation trigger:

Where to?

Public IP address daemon

- Single daemon with public IP address:
  - Management
  - Failover
  - Consistency checking
- Simple management and status CLI
- Simple IP (re)allocation trigger:
  - Simple CLI command: *these nodes* can host addresses

Where to?

Public IP address daemon

- Single daemon with public IP address:
  - Management
  - Failover
  - Consistency checking
- Simple management and status CLI
- Simple IP (re)allocation trigger:
  - Simple CLI command: *these nodes* can host addresses
  - Used as (or in) callback from other daemons when status changes

Where to?

Public IP address daemon

- Single daemon with public IP address:
  - Management
  - Failover
  - Consistency checking
- Simple management and status CLI
- Simple IP (re)allocation trigger:
  - Simple CLI command: *these nodes* can host addresses
  - Used as (or in) callback from other daemons when status changes
- An interface like this should also allow support for LVS, HAProxy, ...

Where to?

Service management daemon

Where to?

Service management daemon

- Four functions:
  - Startup
  - Shutdown

Where to?

Service management daemon

- Four functions:
  - Startup
  - Shutdown
  - Health monitoring
    - Public IP address daemon callback(s) registered to be run on state changes

Where to?

Service management daemon

- Four functions:
  - Startup
  - Shutdown
  - Health monitoring
    - Public IP address daemon callback(s) registered to be run on state changes
  - Reconfiguration when IP addresses change
    - What addresses should services no longer listen on?
    - What addresses should services listen on?

Where to?

Service management daemon

- Four functions:
  - Startup
  - Shutdown
  - Health monitoring
    - Public IP address daemon callback(s) registered to be run on state changes
  - Reconfiguration when IP addresses change
    - What addresses should services no longer listen on?
    - What addresses should services listen on?
- Could we also support something like Pacemaker?

Where to?

Database daemon

Where to?

Database daemon

- After separating everything else, this is what should remain of the current main daemon.

Where to?

Database daemon

- After separating everything else, this is what should remain of the current main daemon.
- The main focus of CTDB

Where to?

Database daemon

- After separating everything else, this is what should remain of the current main daemon.
- The main focus of CTDB
- Functions:
  - Database operations
  - Recovery
  - Vacuuming (garbage collection)

Where to?

Messaging

Where to?

Messaging

- Scalable messaging with multiple daemons across multiple nodes

Where to?

Messaging

- Scalable messaging with multiple daemons across multiple nodes
- Using Samba's Unix domain datagram sockets

Where to?

Messaging

- Scalable messaging with multiple daemons across multiple nodes
- Using Samba's Unix domain datagram sockets
  - Avoids establishing a connection

Where to?

Messaging

- Scalable messaging with multiple daemons across multiple nodes
- Using Samba's Unix domain datagram sockets
  - Avoids establishing a connection
  - Each daemon has to listen only on a single socket

Where to?

Messaging

- Scalable messaging with multiple daemons across multiple nodes
- Using Samba's Unix domain datagram sockets
  - Avoids establishing a connection
  - Each daemon has to listen only on a single socket
  - Need to find sender's socket to send reply

Where to?

Messaging

- Scalable messaging with multiple daemons across multiple nodes
- Using Samba's Unix domain datagram sockets
  - Avoids establishing a connection
  - Each daemon has to listen only on a single socket
  - Need to find sender's socket to send reply
- How to identify a specific deamon / process on a specific node?

How do we get there?

How do we get there?

Itch to re-design **everything**

How do we get there?

Itch to re-design **everything**

- Every new developer's approach . . .

How do we get there?

Itch to re-design **everything**

- Every new developer's approach ...
- Some problems can be designed away

How do we get there?

Itch to re-design **everything**

- Every new developer's approach ...
- Some problems can be designed away
- Daunting task to ensure no knowledge is lost (e.g. database vacuuming and recovery interactions)

How do we get there?

Itch to re-design **everything**

- Every new developer's approach ...
- Some problems can be designed away
- Daunting task to ensure no knowledge is lost (e.g. database vacuuming and recovery interactions)

Freizeit?

- We don't have unlimited people and time...

How do we get there?

Itch to re-design **everything**

- Every new developer's approach . . .
- Some problems can be designed away
- Daunting task to ensure no knowledge is lost (e.g. database vacuuming and recovery interactions)

Freizeit?

- We don't have unlimited people and time. . .
- . . . so this will have to be an incremental effort

How do we get there?

Itch to re-design **everything**

- Every new developer's approach . . .
- Some problems can be designed away
- Daunting task to ensure no knowledge is lost (e.g. database vacuuming and recovery interactions)

Freizeit?

- We don't have unlimited people and time. . .
- . . . so this will have to be an incremental effort
- What can we do to support incremental development?

How do we get there?

New CTDB CLI

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once
- Decide on new command structure (e.g. `ip` or `virsh`)

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once
- Decide on new command structure (e.g. `ip` or `virsh`)
  - e.g. `ctdb ip connection add 10.0.2.132:2049 10.0.1.33:987`

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once
- Decide on new command structure (e.g. `ip` or `virsh`)
  - e.g. `ctdb ip connection add 10.0.2.132:2049 10.0.1.33:987`
  - If command omitted then read list of commands from stdin:

```
$ ctdb ip connection
add 10.0.2.132:2049 10.0.1.33:987
del 10.0.2.133:2049 10.0.1.31:986
```

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once
- Decide on new command structure (e.g. `ip` or `virsh`)
  - e.g. `ctdb ip connection add 10.0.2.132:2049 10.0.1.33:987`
  - If command omitted then read list of commands from stdin:

```
$ ctdb ip connection
add 10.0.2.132:2049 10.0.1.33:987
del 10.0.2.133:2049 10.0.1.31:986
```

  - If arguments omitted then read list of arguments from stdin:

```
$ ctdb ip connection add
10.0.2.132:2049 10.0.1.33:987
10.0.2.133:2049 10.0.1.31:986
```

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once
- Decide on new command structure (e.g. `ip` or `virsh`)
  - e.g. `ctdb ip connection add 10.0.2.132:2049 10.0.1.33:987`
  - If command omitted then read list of commands from stdin:

```
$ ctdb ip connection
add 10.0.2.132:2049 10.0.1.33:987
del 10.0.2.133:2049 10.0.1.31:986
```

  - If arguments omitted then read list of arguments from stdin:

```
$ ctdb ip connection add
10.0.2.132:2049 10.0.1.33:987
10.0.2.133:2049 10.0.1.31:986
```

  - Use readline (or similar) when interactive?

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once
- Decide on new command structure (e.g. `ip` or `virsh`)
  - e.g. `ctdb ip connection add 10.0.2.132:2049 10.0.1.33:987`
  - If command omitted then read list of commands from stdin:

```
$ ctdb ip connection
add 10.0.2.132:2049 10.0.1.33:987
del 10.0.2.133:2049 10.0.1.31:986
```

  - If arguments omitted then read list of arguments from stdin:

```
$ ctdb ip connection add
10.0.2.132:2049 10.0.1.33:987
10.0.2.133:2049 10.0.1.31:986
```

  - Use readline (or similar) when interactive?
- First wrap the current implementation!

How do we get there?

New CTDB CLI

- First decide how users will interact with CTDB
  - Only break user interaction once
- Decide on new command structure (e.g. `ip` or `virsh`)
  - e.g. `ctdb ip connection add 10.0.2.132:2049 10.0.1.33:987`
  - If command omitted then read list of commands from stdin:

```
$ ctdb ip connection
add 10.0.2.132:2049 10.0.1.33:987
del 10.0.2.133:2049 10.0.1.31:986
```

  - If arguments omitted then read list of arguments from stdin:

```
$ ctdb ip connection add
10.0.2.132:2049 10.0.1.33:987
10.0.2.133:2049 10.0.1.31:986
```

  - Use readline (or similar) when interactive?
- First wrap the current implementation!
- Talk to new daemons as they are implemented

How do we get there?

Configuration location and format

How do we get there?

Configuration location and format

- Once again, we should break this (only) once!

How do we get there?

Configuration location and format

- Once again, we should break this (only) once!
- Modular configuration?

How do we get there?

Configuration location and format

- Once again, we should break this (only) once!
- Modular configuration?
- .conf file format?

How do we get there?

Tunnel new protocol over old

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:
  - Attach to `ctdbd` as client (like current recovery daemon)

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:
  - Attach to `ctdbd` as client (like current recovery daemon)
  - Attach to new daemons as peer, talking new protocol

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:
  - Attach to `ctdbd` as client (like current recovery daemon)
  - Attach to new daemons as peer, talking new protocol
  - Wrap new packets in `CTDB_SRVID_TUNNEL_NEW_PROTOCOL` ...

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:
  - Attach to `ctdbd` as client (like current recovery daemon)
  - Attach to new daemons as peer, talking new protocol
  - Wrap new packets in `CTDB_SRVID_TUNNEL_NEW_PROTOCOL` ...
  - ...send to remote node, unwrap, forward to new daemon

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:
  - Attach to `ctdbd` as client (like current recovery daemon)
  - Attach to new daemons as peer, talking new protocol
  - Wrap new packets in `CTDB_SRVID_TUNNEL_NEW_PROTOCOL` ...
  - ...send to remote node, unwrap, forward to new daemon

Really?

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:
  - Attach to `ctdbd` as client (like current recovery daemon)
  - Attach to new daemons as peer, talking new protocol
  - Wrap new packets in `CTDB_SRVID_TUNNEL_NEW_PROTOCOL` ...
  - ...send to remote node, unwrap, forward to new daemon

Really?

- This will really handle all the cases?

How do we get there?

Tunnel new protocol over old

- Start splitting things out as soon as possible!
- Need inter-node transport for new daemons
- Currently `ctdbd` handles all messaging...
- `CTDB_SRVID_TUNNEL_NEW_PROTOCOL`
- Proxy:
  - Attach to `ctdbd` as client (like current recovery daemon)
  - Attach to new daemons as peer, talking new protocol
  - Wrap new packets in `CTDB_SRVID_TUNNEL_NEW_PROTOCOL` ...
  - ...send to remote node, unwrap, forward to new daemon

Really?

- This will really handle all the cases?
- No, but it will minimise the amount of protocol translation...

How do we get there?

Loose coupling via callbacks

How do we get there?

Loose coupling via callbacks

- Daemons don't need to know everything about each other

How do we get there?

Loose coupling via callbacks

- Daemons don't need to know everything about each other
- Most of the inter-daemon events don't happen often

How do we get there?

Loose coupling via callbacks

- Daemons don't need to know everything about each other
- Most of the inter-daemon events don't happen often
- Performance not critical

How do we get there?

Loose coupling via callbacks

- Daemons don't need to know everything about each other
- Most of the inter-daemon events don't happen often
- Performance not critical
- Simplest callback mechanism would be to execute an external program

How do we get there?

Callback for unhealthy node state change

How do we get there?

Callback for unhealthy node state change

1. Service management daemon runs IP daemon's "reallocate IP addresses" callback

How do we get there?

Callback for unhealthy node state change

1. Service management daemon runs IP daemon's "reallocate IP addresses" callback
2. Callback script gathers node states (active/inactive, enabled/disabled), calculates which nodes should host public IP addresses

How do we get there?

Callback for unhealthy node state change

1. Service management daemon runs IP daemon's "reallocate IP addresses" callback
2. Callback script gathers node states (active/inactive, enabled/disabled), calculates which nodes should host public IP addresses
3. Callback script invokes IP daemon CLI, sends list of nodes that can host IP addresses

How do we get there?

Callback for unhealthy node state change

1. Service management daemon runs IP daemon's "reallocate IP addresses" callback
2. Callback script gathers node states (active/inactive, enabled/disabled), calculates which nodes should host public IP addresses
3. Callback script invokes IP daemon CLI, sends list of nodes that can host IP addresses

Put some smarts into the callback scripts

How do we get there?

Callback for unhealthy node state change

1. Service management daemon runs IP daemon's "reallocate IP addresses" callback
2. Callback script gathers node states (active/inactive, enabled/disabled), calculates which nodes should host public IP addresses
3. Callback script invokes IP daemon CLI, sends list of nodes that can host IP addresses

Put some smarts into the callback scripts

- Instead of putting corner cases into daemons and complicating the code (e.g. NoIPTakeoverOnAllDisabled) ...

How do we get there?

Callback for unhealthy node state change

1. Service management daemon runs IP daemon's "reallocate IP addresses" callback
2. Callback script gathers node states (active/inactive, enabled/disabled), calculates which nodes should host public IP addresses
3. Callback script invokes IP daemon CLI, sends list of nodes that can host IP addresses

Put some smarts into the callback scripts

- Instead of putting corner cases into daemons and complicating the code (e.g. NoIPTakeoverOnAllDisabled) ...
- ... keep the daemons as simple as possible and handle some of the corner cases in the callback scripts

How do we get there?

Callbacks for active/inactive node state changes

How do we get there?

Callbacks for active/inactive node state changes

1. Cluster manager runs “maybe recover” callback for database daemon, passes updated list of active nodes

How do we get there?

Callbacks for active/inactive node state changes

1. Cluster manager runs "maybe recover" callback for database daemon, passes updated list of active nodes
2. Database daemon recovers databases

How do we get there?

Callbacks for active/inactive node state changes

1. Cluster manager runs "maybe recover" callback for database daemon, passes updated list of active nodes
2. Database daemon recovers databases
3. Database daemon runs IP daemon's "reallocate IP addresses" callback

How do we get there?

Callbacks for active/inactive node state changes

1. Cluster manager runs "maybe recover" callback for database daemon, passes updated list of active nodes
2. Database daemon recovers databases
3. Database daemon runs IP daemon's "reallocate IP addresses" callback
4. Callback script gathers node states (active/inactive, enabled/disabled), calculates which nodes should host public IP addresses

How do we get there?

Callbacks for active/inactive node state changes

1. Cluster manager runs "maybe recover" callback for database daemon, passes updated list of active nodes
2. Database daemon recovers databases
3. Database daemon runs IP daemon's "reallocate IP addresses" callback
4. Callback script gathers node states (active/inactive, enabled/disabled), calculates which nodes should host public IP addresses
5. Callback script invokes IP daemon CLI, sends list of nodes that can host IP addresses

How do we get there?

Event daemon

How do we get there?

Event daemon

- In the longer term, each daemon could just run its own event scripts via an event library

How do we get there?

Event daemon

- In the longer term, each daemon could just run its own event scripts via an event library
- That's a big step due to current mix of events

How do we get there?

Event daemon

- In the longer term, each daemon could just run its own event scripts via an event library
- That's a big step due to current mix of events
- So:

How do we get there?

Event daemon

- In the longer term, each daemon could just run its own event scripts via an event library
- That's a big step due to current mix of events
- So:
  1. Add an event daemon

How do we get there?

Event daemon

- In the longer term, each daemon could just run its own event scripts via an event library
- That's a big step due to current mix of events
- So:
  1. Add an event daemon
  2. Have it handle all the current events, unchanged

How do we get there?

Event daemon

- In the longer term, each daemon could just run its own event scripts via an event library
- That's a big step due to current mix of events
- So:
  1. Add an event daemon
  2. Have it handle all the current events, unchanged
  3. Split the events and run an event daemon per daemon

How do we get there?

Event daemon

- In the longer term, each daemon could just run its own event scripts via an event library
- That's a big step due to current mix of events
- So:
  1. Add an event daemon
  2. Have it handle all the current events, unchanged
  3. Split the events and run an event daemon per daemon
  4. Perhaps convert to an event library instead of a separate daemon

How do we get there?

Protocol

How do we get there?

Protocol

- Currently structs on the wire

How do we get there?

Protocol

- Currently structs on the wire
- Add abstraction...

How do we get there?

Protocol

- Currently structs on the wire
- Add abstraction...
- ...but still put the same structs on the wire

How do we get there?

Protocol

- Currently structs on the wire
- Add abstraction...
- ...but still put the same structs on the wire
- Then "pull out the tablecloth"!

How do we get there?

Protocol

- Currently structs on the wire
- Add abstraction...
- ...but still put the same structs on the wire
- Then "pull out the tablecloth"!
- Well defined protocol, using XDR (or similar)

How do we get there?

Protocol(s)? . . .

How do we get there?

Protocol(s)? ...

- The database daemon needs to be high performance

How do we get there?

Protocol(s)? ...

- The database daemon needs to be high performance
- Other daemons don't

How do we get there?

Protocol(s)? ...

- The database daemon needs to be high performance
- Other daemons don't
- Quick prototyping?

How do we get there?

Protocol(s)? ...

- The database daemon needs to be high performance
- Other daemons don't
- Quick prototyping?
- RESTful API?

How do we get there?

Protocol(s)? ...

- The database daemon needs to be high performance
- Other daemons don't
- Quick prototyping?
- RESTful API?
- JSON?

How do we get there?

Protocol(s)? ...

- The database daemon needs to be high performance
- Other daemons don't
- Quick prototyping?
- RESTful API?
- JSON?
- Python?

How do we get there?

What else?

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?