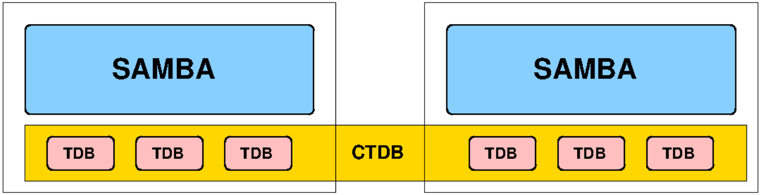
A methodical makeover for CTDB

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

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

Functionality and current architecture

What does CTDB do?



Functionality and current architecture

Functionality

Functionality and current architecture

Functionality

- Cluster membership and leadership

Functionality and current architecture

Functionality

- Cluster membership and leadership
- Cluster database and database recovery

Functionality and current architecture

Functionality

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

Functionality and current architecture

Functionality

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

Functionality and current architecture

Functionality

- 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

Functionality and current architecture

Functionality

- 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
- Logging

Functionality and current architecture

Current architecture

Functionality and current architecture

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

Functionality and current architecture

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

- Main daemon
- Recovery daemon
- Logging daemon

Functionality and current architecture

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

- Main daemon
- Recovery daemon
- Logging daemon

CTDB processes

Ephemeral processes to avoid blocking the main daemon

Functionality and current architecture

Current architecture

CTDB daemons

Processes that exist for the lifetime of CTDB

- Main daemon
- Recovery daemon
- Logging 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

Functionality and current architecture

Mapping function to daemon

Functionality and current architecture

Mapping function to daemon

Main daemon

Recovery daemon

Logging daemon

Functionality and current architecture

Mapping function to daemon

Main daemon

- **Cluster membership**

Recovery daemon

- **Cluster leadership**

Logging daemon

Functionality and current architecture

Mapping function to daemon

Main daemon

- Cluster membership
- **Cluster database access**

Recovery daemon

- Cluster leadership
- **Cluster database recovery**

Logging daemon

Functionality and current architecture

Mapping function to daemon

Main daemon

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

Recovery daemon

- Cluster leadership
- Cluster database recovery

Logging daemon

Functionality and current architecture

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**

Logging daemon

Functionality and current architecture

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

Logging daemon

Functionality and current architecture

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

Logging daemon

- **Logging** :-)

Why makeover?

Why makeover?

- It's time
  - Not a proof of concept anymore . . .

Why makeover?

- It's time
  - Not a proof of concept anymore . . .
- Limitations imposed by design and implementation

Why makeover?

- It's time
    - Not a proof of concept anymore . . .
- Limitations imposed by design and implementation
- Organic Growth
    - Hacks and band-aids

Why makeover?

- It's time
  - Not a proof of concept anymore . . .
- Limitations imposed by design and implementation
- Organic Growth
  - Hacks and band-aids
- Re-factoring?

Why makeover?

- It's time
  - Not a proof of concept anymore . . .
- Limitations imposed by design and implementation
- Organic Growth
  - Hacks and band-aids
- Re-factoring?
  - Easy way to introduce new abstractions (e.g. message lists, locking)

Why makeover?

- It's time
  - Not a proof of concept anymore . . .
- Limitations imposed by design and implementation
- Organic Growth
  - Hacks and band-aids
- Re-factoring?
  - Easy way to introduce new abstractions (e.g. message lists, locking)
  - Can be challenging (e.g. protocol code in CTDB/Samba)

Why makeover?

- It's time
  - Not a proof of concept anymore ...
- Limitations imposed by design and implementation
- Organic Growth
  - Hacks and band-aids
- Re-factoring?
  - Easy way to introduce new abstractions (e.g. message lists, locking)
  - Can be challenging (e.g. protocol code in CTDB/Samba)
- Itch to re-design **everything**
  - Every new developer's approach ...

Why makeover?

- It's time
  - Not a proof of concept anymore . . .
- Limitations imposed by design and implementation
- Organic Growth
  - Hacks and band-aids
- Re-factoring?
  - Easy way to introduce new abstractions (e.g. message lists, locking)
  - Can be challenging (e.g. protocol code in CTDB/Samba)
- Itch to re-design **everything**
  - Every new developer's approach . . .
  - Some problems can be designed away

Why makeover?

- It's time
  - Not a proof of concept anymore . . .
- Limitations imposed by design and implementation
- Organic Growth
  - Hacks and band-aids
- Re-factoring?
  - Easy way to introduce new abstractions (e.g. message lists, locking)
  - Can be challenging (e.g. protocol code in CTDB/Samba)
- 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)

Limitations: Design

Limitations: Design

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

Limitations: Design

- 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

Limitations: Design

- 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

Limitations: Design

- 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!

Limitations: Design

- 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

Limitations: Design

- 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

Limitations: Design

- 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
- Centralised state
  - Some state is in main daemon but is used in recovery daemon
- Tight coupling
  - Membership, service health, IP allocation are tightly coupled

Limitations: Implementation

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

Limitations: Implementation

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

Limitations: Implementation

- 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)

Limitations: Implementation

- 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)

Limitations: Implementation

- 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

Limitations: Implementation

- 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

Limitations: Implementation

- 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
- Unstructured CLI and configuration

Limitations: Implementation

- 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
- Unstructured CLI and configuration

Need to re-design

- Scalability, Maintainability

Component: Logging daemon

Motivation

What is the smallest chunk that can be split as a separate daemon?

Component: Logging daemon

Motivation

What is the smallest chunk that can be split as a separate daemon?

Logging daemon

- Self-contained code
- Can be used as a template for other daemons
- Looks simple enough. . .

Component: Logging daemon

Before: Custom logging daemon

Component: Logging daemon

Before: Custom logging daemon

Why?	`syslog(3)` blocks when syslog daemon gets busy
What?	Log each received message using `syslog(3)`
How?	Custom UDP protocol

Component: Logging daemon

Before: Custom logging daemon

Why?	`syslog(3)` blocks when syslog daemon gets busy
What?	Log each received message using `syslog(3)`
How?	Custom UDP protocol

Problems

Component: Logging daemon

Before: Custom logging daemon

Why?	`syslog(3)` blocks when syslog daemon gets busy
What?	Log each received message using `syslog(3)`
How?	Custom UDP protocol

Problems

- Only used when syslog enabled, not file logging
- File logging can block too!

Component: Logging daemon

Before: Custom logging daemon

Why?	`syslog(3)` blocks when syslog daemon gets busy
What?	Log each received message using `syslog(3)`
How?	Custom UDP protocol

Problems

- Only used when syslog enabled, not file logging
- File logging can block too!
- Protocol is “structs on the wire”

Component: Logging daemon

Before: Custom logging daemon

Why?	`syslog(3)` blocks when syslog daemon gets busy
What?	Log each received message using `syslog(3)`
How?	Custom UDP protocol

Problems

- Only used when syslog enabled, not file logging
- File logging can block too!
- Protocol is “structs on the wire”

After?

Component: Logging daemon

Before: Custom logging daemon

Why?	`syslog(3)` blocks when syslog daemon gets busy
What?	Log each received message using `syslog(3)`
How?	Custom UDP protocol

Problems

- Only used when syslog enabled, not file logging
- File logging can block too!
- Protocol is “structs on the wire”

After?

- Shiny new daemon with well-defined protocol. . .

Component: Logging daemon

Before: Custom logging daemon

Why?	`syslog(3)` blocks when syslog daemon gets busy
What?	Log each received message using `syslog(3)`
How?	Custom UDP protocol

Problems

- Only used when syslog enabled, not file logging
- File logging can block too!
- Protocol is “structs on the wire”

After?

- Shiny new daemon with well-defined protocol. . .
- . . . that handles all logging

Component: Logging daemon

The big idea!

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!
- Define **protocol** and auto-generate marshalling code!

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!
- Define **protocol** and auto-generate marshalling code!
- Use all this to write logging daemon (as a template)!

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!
- Define **protocol** and auto-generate marshalling code!
- Use all this to write logging daemon (as a template)!
- And then use the template for writing other daemons!

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!
- Define **protocol** and auto-generate marshalling code!
- Use all this to write logging daemon (as a template)!
- And then use the template for writing other daemons!

The big problem!

- Logging is hard!
- How do you handle errors in logging daemon?

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!
- Define **protocol** and auto-generate marshalling code!
- Use all this to write logging daemon (as a template)!
- And then use the template for writing other daemons!

The big problem!

- Logging is hard!
- How do you handle errors in logging daemon?

The better idea!

- We're not in the logging business . . . daemons already exist!

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!
- Define **protocol** and auto-generate marshalling code!
- Use all this to write logging daemon (as a template)!
- And then use the template for writing other daemons!

The big problem!

- Logging is hard!
- How do you handle errors in logging daemon?

The better idea!

- We're not in the logging business . . . daemons already exist!
- Use RFC5424 message format

Component: Logging daemon

The big idea!

- Create an asynchronous framework for CTDB daemons!
- Use Samba's `tevent_req` framework!
- Define **protocol** and auto-generate marshalling code!
- Use all this to write logging daemon (as a template)!
- And then use the template for writing other daemons!

The big problem!

- Logging is hard!
- How do you handle errors in logging daemon?

The better idea!

- We're not in the logging business ... daemons already exist!
- Use RFC5424 message format
- Transmit via UDP as per RFC5426

Component: Logging daemon

So, how did that work out?

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because...
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?
- `rsyslogd` doesn't speak RFC5424 on Unix domain socket?

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?
- `rsyslogd` doesn't speak RFC5424 on Unix domain socket?
- Learn about RFC3164!

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?
- `rsyslogd` doesn't speak RFC5424 on Unix domain socket?
- Learn about RFC3164!
- Location of socket is not standardised

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?
- `rsyslogd` doesn't speak RFC5424 on Unix domain socket?
- Learn about RFC3164!
- Location of socket is not standardised
- Much of RFC3164 is only recommended. . .

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?
- `rsyslogd` doesn't speak RFC5424 on Unix domain socket?
- Learn about RFC3164!
- Location of socket is not standardised
- Much of RFC3164 is only recommended. . .
- . . . and sometimes not supported

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?
- `rsyslogd` doesn't speak RFC5424 on Unix domain socket?
- Learn about RFC3164!
- Location of socket is not standardised
- Much of RFC3164 is only recommended. . .
- . . . and sometimes not supported
- FreeBSD supports RFC3164, not RFC5424, over UDP

Component: Logging daemon

So, how did that work out?

- First Linux version quite easy but not merged because. . .
- Unified Samba/CTDB build coming up (see later)
- Samba's `debug.{ch}` is completely different to CTDB's
- Spend a month completing the unified build
- Send to Unix domain socket in non-blocking mode?
- `rsyslogd` doesn't speak RFC5424 on Unix domain socket?
- Learn about RFC3164!
- Location of socket is not standardised
- Much of RFC3164 is only recommended. . .
- . . . and sometimes not supported
- FreeBSD supports RFC3164, not RFC5424, over UDP
- Tear out hair. . .

Component: Logging daemon

CTDB `logging=syslog*` options

`syslog`	Use `syslog(3)`
`syslog:nonblocking`	RFC3164 to Unix domain socket
`syslog:udp`	RFC3164 to UDP socket
`syslog:udp-rfc5424`	RFC5424 to UDP socket (RFC5426)

Component: Logging daemon

CTDB `logging=syslog*` options

`syslog`	Use `syslog(3)`
`syslog:nonblocking`	RFC3164 to Unix domain socket
`syslog:udp`	RFC3164 to UDP socket
`syslog:udp-rfc5424`	RFC5424 to UDP socket (RFC5426)

After

Component: Logging daemon

CTDB `logging=syslog*` options

`syslog`	Use `syslog(3)`
`syslog:nonblocking`	RFC3164 to Unix domain socket
`syslog:udp`	RFC3164 to UDP socket
`syslog:udp-rfc5424`	RFC5424 to UDP socket (RFC5426)

After

- A lot of time passed. . . more than 12 months

Component: Logging daemon

CTDB `logging=syslog*` options

`syslog`	Use `syslog(3)`
`syslog:nonblocking`	RFC3164 to Unix domain socket
`syslog:udp`	RFC3164 to UDP socket
`syslog:udp-rfc5424`	RFC5424 to UDP socket (RFC5426)

After

- A lot of time passed. . . more than 12 months
- Above merged into (Samba) master branch

Component: Logging daemon

CTDB `logging=syslog*` options

`syslog`	Use `syslog(3)`
`syslog:nonblocking`	RFC3164 to Unix domain socket
`syslog:udp`	RFC3164 to UDP socket
`syslog:udp-rfc5424`	RFC5424 to UDP socket (RFC5426)

After

- A lot of time passed. . . more than 12 months
- Above merged into (Samba) master branch
- Retired from the logging business

Component: Logging daemon

CTDB `logging=syslog*` options

`syslog`	Use `syslog(3)`
`syslog:nonblocking`	RFC3164 to Unix domain socket
`syslog:udp`	RFC3164 to UDP socket
`syslog:udp-rfc5424`	RFC5424 to UDP socket (RFC5426)

After

- A lot of time passed. . . more than 12 months
- Above merged into (Samba) master branch
- Retired from the logging business

Future?

- Promote some of this to Samba's `debug.{ch}`

New design

Motivation

Separate functionality in individual daemons

New design

Motivation

Separate functionality in individual daemons

Design

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

New design: Public IP address daemon

New design: Public IP address daemon

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

New design: Public IP address daemon

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

New design: Public IP address daemon

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

New design: 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

New design: 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
  - Callback from other daemons when status changes

New design: 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
  - Callback from other daemons when status changes
  - Callback can be a script that gathers extra status data.
    For example, cluster membership and/or service health status.

New design: 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
  - Callback from other daemons when status changes
  - Callback can be a script that gathers extra status data.
    For example, cluster membership and/or service health status.
- An interface like this should also allow support for LVS, HAProxy, . . .

New design: Service management daemon

- Four functions:

New design: Service management daemon

- Four functions:
  - Startup

New design: Service management daemon

- Four functions:
  - Startup
  - Shutdown

New design: Service management daemon

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

New design: 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?

New design: 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?

New design: Cluster management daemon

New design: Cluster management daemon

- Membership:

Connected	according to heartbeat or similar
Active	if not banned, administratively stopped

New design: Cluster management daemon

- Membership:

Connected	according to heartbeat or similar
Active	if not banned, administratively stopped

- Leadership
  - Coordinates database recovery
  - Coordinates public IP address (re)allocation

New design: Cluster management daemon

- Membership:

Connected	according to heartbeat or similar
Active	if not banned, administratively stopped

- Leadership
  - Coordinates database recovery
  - Coordinates public IP address (re)allocation
- Callbacks registered for state changes

New design: Cluster management daemon

- Membership:

Connected	according to heartbeat or similar
Active	if not banned, administratively stopped

- Leadership
  - Coordinates database recovery
  - Coordinates public IP address (re)allocation
- Callbacks registered for state changes
- Can we support Heartbeat, etcd (or similar) as an alternative?

New design: Database daemon

New design: Database daemon

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

New design: Database daemon

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

New design: 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)

New design: Messaging

New design: Messaging

- Scalable messaging with multiple daemons across multiple nodes

New design: Messaging

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

New design: Messaging

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

New design: 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

New design: 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

New design: 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?

Status

Question

We didn't get all of this done, did we?

Distractions

Distractions

- CTDB

Distractions

- CTDB
    - Framework, experiments with logging daemon, . . .

Distractions

- CTDB
  - Framework, experiments with logging daemon, . . .
  - Unified Samba/CTDB tree and build

Distractions

- CTDB
  - Framework, experiments with logging daemon, . . .
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)

Distractions

- CTDB
  - Framework, experiments with logging daemon, . . .
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling

Distractions

- CTDB
  - Framework, experiments with logging daemon, ...
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support

Distractions

- CTDB
  - Framework, experiments with logging daemon, ...
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster

Distractions

- CTDB
  - Framework, experiments with logging daemon, . . .
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster
  - Create virtual RHEL/CentOS libvirt/KVM clusters. . .

Distractions

- CTDB
  - Framework, experiments with logging daemon, . . .
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster
  - Create virtual RHEL/CentOS libvirt/KVM clusters. . .
  - . . . for testing clustered Samba

Distractions

- CTDB
  - Framework, experiments with logging daemon, ...
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster
  - Create virtual RHEL/CentOS libvirt/KVM clusters...
  - ...for testing clustered Samba
  - Written in `bash(1)` since 2008!

Distractions

- CTDB
  - Framework, experiments with logging daemon, . . .
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster
  - Create virtual RHEL/CentOS libvirt/KVM clusters. . .
  - . . . for testing clustered Samba
  - Written in `bash(1)` since 2008!
  - See LCA2009 presentation with Tridge

Distractions

- CTDB
    - Framework, experiments with logging daemon, ...
    - Unified Samba/CTDB tree and build
    - Portability (Linux on Power, AIX, FreeBSD)
    - Performance: lock scheduling
    - Fix IPv6 support
- Autocluster
    - Create virtual RHEL/CentOS libvirt/KVM clusters...
    - ...for testing clustered Samba
    - Written in `bash(1)` since 2008!
    - See LCA2009 presentation with Tridge
    - RHEL 7 support

Distractions

- CTDB
  - Framework, experiments with logging daemon, . . .
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster
  - Create virtual RHEL/CentOS libvirt/KVM clusters. . .
  - . . . for testing clustered Samba
  - Written in `bash(1)` since 2008!
  - See LCA2009 presentation with Tridge
  - RHEL 7 support
  - Modularisation

Distractions

- CTDB
  - Framework, experiments with logging daemon, ...
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster
  - Create virtual RHEL/CentOS libvirt/KVM clusters...
  - ...for testing clustered Samba
  - Written in `bash(1)` since 2008!
  - See LCA2009 presentation with Tridge
  - RHEL 7 support
  - Modularisation
  - IPv6 support

Distractions

- CTDB
  - Framework, experiments with logging daemon, ...
  - Unified Samba/CTDB tree and build
  - Portability (Linux on Power, AIX, FreeBSD)
  - Performance: lock scheduling
  - Fix IPv6 support
- Autocluster
  - Create virtual RHEL/CentOS libvirt/KVM clusters...
  - ...for testing clustered Samba
  - Written in `bash(1)` since 2008!
  - See LCA2009 presentation with Tridge
  - RHEL 7 support
  - Modularisation
  - IPv6 support
  - `git://git.samba.org/autocluster.git`

But wait, there's more. . .

Well, not a lot more, but a little more. . .

Beginning of a makeover

Beginning of a makeover

- Lots of re-design, lots of work

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?
    - Sounds good, but a huge step to get working code

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?
  - Sounds good, but a huge step to get working code
  - Limited development team

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?
  - Sounds good, but a huge step to get working code
  - Limited development team
- Incremental updates

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?
  - Sounds good, but a huge step to get working code
  - Limited development team
- Incremental updates
  - Harness existing testing infrastructure

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?
    - Sounds good, but a huge step to get working code
    - Limited development team
- Incremental updates
    - Harness existing testing infrastructure
    - Will require throw-away glue code

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?
  - Sounds good, but a huge step to get working code
  - Limited development team
- Incremental updates
  - Harness existing testing infrastructure
  - Will require throw-away glue code
  - Where to start?

Beginning of a makeover

- Lots of re-design, lots of work
- Start with a clean slate?
  - Sounds good, but a huge step to get working code
  - Limited development team
- Incremental updates
  - Harness existing testing infrastructure
  - Will require throw-away glue code
  - Where to start?

Protocol handling

Samba and CTDB have separate implementation of protocol

Makeover: Protocol handling

- Implement libctdb

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?
  - Implemented few messages, but not database operations

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?
  - Implemented few messages, but not database operations
  - Provided mostly synchronous and some asynchronous API

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?
    - Implemented few messages, but not database operations
    - Provided mostly synchronous and some asynchronous API
    - Hard to get thread-safe asynchronous API right

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?
  - Implemented few messages, but not database operations
  - Provided mostly synchronous and some asynchronous API
  - Hard to get thread-safe asynchronous API right
  - No consumers for libctdb (partial use by ctdb CLI)

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?
  - Implemented few messages, but not database operations
  - Provided mostly synchronous and some asynchronous API
  - Hard to get thread-safe asynchronous API right
  - No consumers for libctdb (partial use by ctdb CLI)
- Implement libctdbapi
  - CTDB protocol marshalling API (client and server)

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?
    - Implemented few messages, but not database operations
    - Provided mostly synchronous and some asynchronous API
    - Hard to get thread-safe asynchronous API right
    - No consumers for libctdb (partial use by ctdb CLI)
- Implement libctdbapi
    - CTDB protocol marshalling API (client and server)
    - Rewrite Samba's CTDB interface using libctdbapi

Makeover: Protocol handling

- Implement libctdb
- But wait, wasn't there a libctdb already?
  - Implemented few messages, but not database operations
  - Provided mostly synchronous and some asynchronous API
  - Hard to get thread-safe asynchronous API right
  - No consumers for libctdb (partial use by ctdb CLI)
- Implement libctdbapi
  - CTDB protocol marshalling API (client and server)
  - Rewrite Samba's CTDB interface using libctdbapi
  - Rewrite CTDB server side using libctdb-serverapi?

Makeover: for the rest . . .

Makeover: for the rest . . .

- Keep hacking in the spare time . . .

Makeover: for the rest . . .

- Keep hacking in the spare time . . .
- The pace is too slow to keep up with Samba releases

Makeover: for the rest . . .

- Keep hacking in the spare time . . .
- The pace is too slow to keep up with Samba releases

Better solution

Get smart(er) developers involved!

Legal Statement

Questions?