The CTDB Report

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

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

SambaXP 2019

Overview

- Progress in the past year
- Plans presented at SambaXP 2017/2018
- Design ideas
- New daemons
- Way forward

Progress in the past year

Progress in the past year

Committers

Alexander Bokovoy	4
Amitay Isaacs	82
Andreas Schneider	11
Andrew Bartlett	3
Carlos O'Donell	1
Christof Schmitt	3
David Disseldorp	8
Douglas Bagnall	1
Martin Schwenke	315
Noel Power	4
Olly Betts	1
Rafael David Tinoco	1
Ralph Boehme	1
Ralph Wuerthner	1
Samuel Cabrero	1
Stefan Metzmacher	5
Swen Schillig	17
Volker Lendecke	6
Zhu Shangzhong	1
	466

Progress in the past year

Commits by area

Configuration changes for 4.9	23
Add eventd (including preparation + fixes)	64
Portability	32
Portability - Packet handling	35
Recovery lock reliability	20
Vacuuming improvements	11
Scripts - NFS fixes for systemd	13
Test - `local_daemons.sh`	53
Test - generic improvements	52
Build/WAF 2.0/Py3	22
Generic Samba clean-ups	13
Other	128
	466

Plans presented at SambaXP 2017/2018

Plans presented at SambaXP 2017/2018

Separate daemons

- event daemon
- service daemon
- failover daemon + connection tracking daemon
- cluster daemon
- database daemon
- transport
- smbd proxy

Plans presented at SambaXP 2017/2018

eventd

serviced

failoverd + conntrackd

clusterd + databased

transport

Plans presented at SambaXP 2017/2018

eventd

- In Samba 4.9

serviced

failoverd + conntrackd

clusterd + databased

transport

Plans presented at SambaXP 2017/2018

`eventd`

- In Samba 4.9

`serviced`

- Initial version finished before 4.9

`failoverd` + `conntrackd`

`clusterd` + `databased`

`transport`

Plans presented at SambaXP 2017/2018

eventd

- In Samba 4.9

serviced

- Initial version finished before 4.9

failoverd + conntrackd

- Hurriedly, nearly finished before 4.9
- A lot of copy & paste from serviced
- Could have gone into 4.9...
- ...but required lots of integration work

clusterd + databased

transport

Plans presented at SambaXP 2017/2018

`eventd`

- In Samba 4.9

`serviced`

- Initial version finished before 4.9

`failoverd` + `conntrackd`

- Hurriedly, nearly finished before 4.9
- A lot of copy & paste from `serviced`
- Could have gone into 4.9. . .
- . . . but required lots of integration work

`clusterd` + `databased`

- Not implemented

`transport`

Plans presented at SambaXP 2017/2018

`eventd`

- In Samba 4.9

`serviced`

- Initial version finished before 4.9

`failoverd` + `conntrackd`

- Hurriedly, nearly finished before 4.9
- A lot of copy & paste from `serviced`
- Could have gone into 4.9…
- …but required lots of integration work

`clusterd` + `databased`

- Not implemented

`transport`

- In design phase

Plans presented at SambaXP 2017/2018

Status

Conclusions

Plans presented at SambaXP 2017/2018

Status

- Components not mature enough for 4.9, not merged

Conclusions

Plans presented at SambaXP 2017/2018

Status

- Components not mature enough for 4.9, not merged

Conclusions

- Lots of boilerplate code for each daemon and client tool
- Each daemon with a unix domain socket
- Separate protocol for each daemon
  - Client — Server
  - Server — Server ?

Plans presented at SambaXP 2017/2018

Status

- Components not mature enough for 4.9, not merged

Conclusions

- Lots of boilerplate code for each daemon and client tool
- Each daemon with a unix domain socket
- Separate protocol for each daemon
  - Client — Server
  - Server — Server ?
- `sock_daemon`

Plans presented at SambaXP 2017/2018

Status

- Components not mature enough for 4.9, not merged

Conclusions

- Lots of boilerplate code for each daemon and client tool
- Each daemon with a unix domain socket
- Separate protocol for each daemon
  - Client — Server
  - Server — Server ?
- `sock_daemon`
- Testing becomes easier
  - No need for fake daemons

Plans presented at SambaXP 2017/2018

Status

- Components not mature enough for 4.9, not merged

Conclusions

- Lots of boilerplate code for each daemon and client tool
- Each daemon with a unix domain socket
- Separate protocol for each daemon
  - Client — Server
  - Server — Server ?
- `sock_daemon`
- Testing becomes easier
  - No need for fake daemons
- . . . and complicated
  - `serviced` → `eventd`
  - `failoverd` → `eventd`, `transport`
  - Need multiple daemons for setup

Design ideas

Design ideas

Topics

- Reduce copy/paste code
- Simplify testing
- Unify protocol
- Too many sockets

Design ideas

Reduce copy/paste code

Design ideas

Reduce copy/paste code

- `sock_daemon` was good for abstracting
- Forces boilerplate code for each daemon
- Avoids handling protocol, but not very effective

Design ideas

Reduce copy/paste code

- `sock_daemon` was good for abstracting
- Forces boilerplate code for each daemon
- Avoids handling protocol, but not very effective
- Enter `tdaemon`

Design ideas

Reduce copy/paste code

- `sock_daemon` was good for abstracting
- Forces boilerplate code for each daemon
- Avoids handling protocol, but not very effective
- Enter `tdaemon`
- And possibly `tclient`

Design ideas

Simplify testing

Design ideas

Simplify testing

- Unit testing of ctdb daemon is impossible!
- Separate daemons are easier to unit test
- How to handle dependencies?
- Can we combine multiple daemons?

Design ideas

Simplify testing

- Unit testing of ctdb daemon is impossible!
- Separate daemons are easier to unit test
- How to handle dependencies?
- Can we combine multiple daemons?
- Each daemon is a `tevent_req` computation . . .
- One daemon to rule them all?

Design ideas

Simplify testing

- Unit testing of ctdb daemon is impossible!
- Separate daemons are easier to unit test
- How to handle dependencies?
- Can we combine multiple daemons?
- Each daemon is a `tevent_req` computation . . .
- One daemon to rule them all?
- `masterd`

Design ideas

Unify protocol

Design ideas

Unify protocol

- Each daemon needs some common “controls”
- Should Client — Server be different from Server — Server?
- New protocol?
- Design it right from beginning – endian neutral
- Common “controls” can be implemented once

Design ideas

Unify protocol

- Each daemon needs some common "controls"
- Should Client — Server be different from Server — Server?
- New protocol?
- Design it right from beginning – endian neutral
- Common "controls" can be implemented once
- `tdaemon`

Design ideas

Too many sockets

Design ideas

Too many sockets

- Each daemon with unix domain socket
- Easy to test, . . .
- . . . but gets messy to manage many sockets

Design ideas

Too many sockets

- Each daemon with unix domain socket
- Easy to test, . . .
- . . . but gets messy to manage many sockets
- Messaging server?
- . . . Unix datagram messaging

Design ideas

Too many sockets

- Each daemon with unix domain socket
- Easy to test, . . .
- . . . but gets messy to manage many sockets
- Messaging server?
- . . . Unix datagram messaging
- `transportd`
- Every daemon now uses common `transport` client code

Design ideas

Too many sockets

- Each daemon with unix domain socket
- Easy to test, . . .
- . . . but gets messy to manage many sockets
- Messaging server?
- . . . Unix datagram messaging
- `transportd`
- Every daemon now uses common `transport` client code
- Works very well for `tdaemon` abstraction

New daemons

New daemon

Topics

- Master daemon
- Transport daemon

New daemons

Master daemon

New daemons

Master daemon

- Start and monitor multiple daemons

New daemons

Master daemon

- Start and monitor multiple daemons
  - Multiple process model
  - Single process model

New daemons

Master daemon

- Start and monitor multiple daemons
  - Multiple process model
  - Single process model
- Bundle all dependencies for testing in one daemon

New daemons

Master daemon

- Start and monitor multiple daemons
    - Multiple process model
    - Single process model
- Bundle all dependencies for testing in one daemon
- No, this is not `systemd` :-)

New daemons

Transport daemon

New daemons

Transport daemon

- All daemons talk to transport
- Routes packets between daemons
- Routes packets between nodes
- Understands just enough protocol for routing
- Keep it light and blazing fast!

New daemons

Transport daemon

- All daemons talk to transport
- Routes packets between daemons
- Routes packets between nodes
- Understands just enough protocol for routing
- Keep it light and blazing fast!
- Further ideas
  - Minimise dynamic memory allocation…
  - …to zero?

Way forward

Way forward

Incremental development

Way forward

Incremental development

- To make best long-term progress, avoid churn

Way forward

Incremental development

- To make best long-term progress, avoid churn
- To avoid churn, we need to develop against `transportd` API

Way forward

Incremental development

- To make best long-term progress, avoid churn
- To avoid churn, we need to develop against `transportd` API
- Either need to develop new database daemon against `transportd` API. . .

Way forward

Incremental development

- To make best long-term progress, avoid churn
- To avoid churn, we need to develop against `transportd` API
- Either need to develop new database daemon against `transportd` API. . .
- . . . or retrofit existing `ctdbd` against `transportd` API

Way forward

Incremental development

- To make best long-term progress, avoid churn
- To avoid churn, we need to develop against `transportd` API
- Either need to develop new database daemon against `transportd` API…
- …or retrofit existing `ctdbd` against `transportd` API
- The latter involves significant churn

Way forward

Fake `transportd` client

Way forward

Fake `transportd` client

- Implement alternative `transportd` client code that uses current `ctdbd` as transport?

Way forward

Fake transportd client

- Implement alternative transportd client code that uses current ctdbd as transport?
- Implement new components using this API

Way forward

Fake transportd client

- Implement alternative transportd client code that uses current ctdbd as transport?
- Implement new components using this API
- Implement new database daemon and transportd

Way forward

Fake `transportd` client

- Implement alternative `transportd` client code that uses current `ctdbd` as transport?
- Implement new components using this API
- Implement new database daemon and `transportd`
- However, first step involves churn

Way forward

Recovery scalability

Way forward

Recovery scalability

- Recovery master node is (probably) a bottleneck for recovery

Way forward

Recovery scalability

- Recovery master node is (probably) a bottleneck for recovery
- Recovery master could distribute recovery of individual databases across nodes

Way forward

Recovery scalability

- Recovery master node is (probably) a bottleneck for recovery
- Recovery master could distribute recovery of individual databases across nodes
- Could implement in current code

Way forward

Recovery scalability

- Recovery master node is (probably) a bottleneck for recovery
- Recovery master could distribute recovery of individual databases across nodes
- Could implement in current code
- Churn!

Way forward

Problem

- Every time we churn we delay progress towards new design. . .

Way forward

CTDB developers needed

- Samba Team has one full time CTDB developer

Way forward

CTDB developers needed

- Samba Team has one full time CTDB developer
- Some amount of burnout. . .

Way forward

CTDB developers needed

- Samba Team has one full time CTDB developer
- Some amount of burnout. . .
- Any volunteers?

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?