Ringpop Documentation

Release 0.1.0

Uber Technologies, Inc.

October 01, 2015

1 Getting Started **3**
1.1 What is Ringpop? . . . 3
1.2 Why use Ringpop? . . . 3
1.3 Installation . . . 4

2 Running Ringpop **5**
2.1 Running with tick-cluster . . . 5
2.2 Running from the command-line . . . 5
2.3 Administration . . . 5
2.4 Configuration . . . 6
2.5 Deploying . . . 6
2.6 Monitoring . . . 6
2.7 Benchmarks . . . 6
2.8 Troubleshooting . . . 6

3 Programming Ringpop **7**
3.1 Code Walkthrough . . . 7
3.2 API . . . 8
3.3 An Example Express App . . . 10

4 Architecture, Design, and Implementation **13**
4.1 Concepts . . . 13
4.2 How Ringpop Works . . . 14
4.3 Extensions . . . 16

5 References **19**
5.1 FAQ . . . 19
5.2 Glossary . . . 19
5.3 Use Cases . . . 21
5.4 Papers . . . 21
5.5 Presentations . . . 22

6 Community **23**
6.1 Google Group . . . 23
6.2 Contributing . . . 23
6.3 License . . . 23

Ringpop is a library that brings cooperation and coordination to distributed applications.

Interested? Start with the Getting Started section below.

CHAPTER 1

Getting Started

Start here to learn about Ringpop and how to install it.

1.1 What is Ringpop?

Ringpop is a library that brings cooperation and coordination to applications that would otherwise run as a set of independent worker processes.

Ringpop implements a membership protocol that allows those workers to discover one another and use the communication channels established between them as a means of disseminating information, detecting failure, and ultimately converging on a consistent membership list. Consistent hashing is then applied on top of that list and gives an application the ability to define predictable behavior and data storage facilities within a custom keyspace. The keyspace is partitioned and evenly assigned to the individual instances of an application. Clients of the application remain simple and need not know of the underlying cooperation between workers nor chosen partitioning scheme. A request can be sent to any instance, and Ringpop intelligently forwards the request to the "correct" instance as defined by a hash ring lookup.

Ringpop makes it possible to build extremely scalable and fault-tolerant distributed systems with 3 main capabilities: membership protocol, consistent hashing, and forwarding.

1.2 Why use Ringpop?

Ringpop is a library that brings application-layer sharding to your services, partitioning data in a way that is fault-tolerant and scalable. Ringpop offers availability over consistency, and massive scaling capability for architectures with realtime, highly transactional and volatile data.

Ringpop delivers resiliency to your services for complex and scaling architectures while keeping operational overhead low. Traditional environments have a number of stateless servers connected to a database that stores state information. With Ringpop, state is not stored in the database; servers hold the state in memory.

Typically, sharding is client aware. With Ringpop, sharding is client agnostic, and done under the covers by providing a simple hash ring abstraction. Ringpop introduces sharding into the application layer rather than the data store, allowing for a highly available, high performant and scalable infrastructure with systems that are self-healing. Ringpop allows applications to distribute state, maintaining a hash ring of each service instance in an application cluster. It automatically detects failures and other changes to the state of the ring through the SWIM gossip protocol. Any added capacity is easily integrated into an existing Ringpop cluster, and traffic gets evenly distributed over the new capacity, helping to avoid downtime.

1.3 Installation

```
npm install ringpop
```

CHAPTER 2

Running Ringpop

Learn how to incorporate Ringpop into your application.

2.1 Running with tick-cluster

`tick-cluster` is a utility located in the `scripts/` directory of the Ringpop repo that allows you to quickly spin up a Ringpop cluster of arbitrary size and test basic failure modes: suspending, killing and respawning nodes.

To use `tick-cluster`, first clone the repo and install Ringpop's dependencies:

```
$ git clone git@github.com:uber/ringpop.git
$ npm install
```

Then run `tick-cluster`:

```
$ ./scripts/tick-cluster.js [-n size-of-cluster] [-i interpreter-that-runs-program] <ringpo
```

`tick-cluster` will spawn a child process for each node in the cluster. They will bootstrap themselves using an auto-generated `hosts.json` bootstrap file and converge on a single membership list within seconds. Commands can be issued against the cluster while `tick-cluster` runs. Press `h` or `?` to see which commands are available.

Whenever it is specified, the program is run by an interpreter, otherwise the program should be a binary. The cluster size defaults to 5.

Here's a sample of the output you may see after launching a 7-node cluster with `tick-cluster`:

```
$ ./scripts/tick-cluster.js -n 7 -i node ./main.js
[init] 11:11:52.805 tick-cluster started d: debug flags, g: gossip, j: join, k: kill, K: re
[cluster] 11:11:52.807 using 10.80.135.224 to listen
[init] 11:11:52.818 started 7 procs: 76365, 76366, 76367, 76368, 76369, 76370, 76371
```

2.2 Running from the command-line

Content coming soon...

2.3 Administration

Content coming soon...

2.4 Configuration

Content coming soon...

2.5 Deploying

Content coming soon...

2.6 Monitoring

Content coming soon...

2.7 Benchmarks

Content coming soon...

2.8 Troubleshooting

Content coming soon...

CHAPTER 3

Programming Ringpop

You may decide that Ringpop is right for your application and want to know how to program against it. Below you'll find information about how to use Ringpop within your application and the API Ringpop exposes to the application developer.

3.1 Code Walkthrough

The first thing you'll want is a handle on a Ringpop instance. You'll first need an instance of TChannel, the underlying transport for Ringpop:

Node.js

```
var TChannel = require(`TChannel');

var tchannel = new TChannel();
var subChannel = tchannel.makeSubChannel({
    serviceName: `ringpop'
});
```

Then decide on a listening address for TChannel. We'll leave that exercise for the reader. For the purposes of this exercise, let's assume we're using:

Node.js

```
var host = `172.18.27.228';
var port = 3000;
```

You're almost ready for Ringpop. Before we get to it, we'll need a list of addresses that act as the seed for Ringpop to join a cluster of other nodes. Let's assume that there are other Ringpop nodes that are or will be available:

Node.js

```
var bootstrapNodes = [`172.18.27.228:3000', `172.18.27.228:3001',
    `172.18.27.228:3002'];
```

We're there! Instantiate Ringpop:

Node.js

```
var ringpop = new Ringpop({
    app: `yourapp',
    hostPort: host + `:' + port,
    channel: subChannel
});
```

```
ringpop.setupChannel();
ringpop.channel.once(`listening', onListening);
ringpop.channel.listen(port, host);

function onListening() {
    ringpop.bootstrap(bootstrapNodes, onBootstrap);
}

function onBootstrap(err) {
    if (err) {
        // Fatal error
        return;
    }

    // Start listening for application traffic
}
```

When TChannel starts listening for connections, Ringpop is ready to be bootstrapped. Bootstrapping consists of having Ringpop send out a join request to a number of random hosts selected from `bootstrapNodes`. Your application is ready to serve traffic when Ringpop successfully joins a cluster.

As requests arrive, you'll want to lookup a request's key against the ring. If the key hashes to the address of Ringpop (`hostPort`), your request may be handled locally, otherwise it'll need to be forwarded to the correct Ringpop. The typical pattern you'll see looks similar to:

Node.js

```
var destination = ringpop.lookup(key);

if (destination === ringpop.whoami()) {
    // Handle the request
} else {
    // Forward the request
}
```

This pattern has been codified in Ringpop's `handleOrProxy` function as a convenience and can be used to forward HTTP traffic over TChannel. If the request should not be handled locally, `handleOrProxy` will return `false`:

Node.js

```
if (ringpop.handleOrProxy(key, req, res, opts)) {
    // Handle the request
}
```

That's really all there is to using Ringpop within your application. For a deeper dive into all of the other bells and whistles Ringpop has to offer we refer you to the API section below and the Running Ringpop page.

3.2 API

3.2.1 `Ringpop(opts)`

Creates an instance of Ringpop.

- `app` - The title of your application. It is used to protect your application's ring from cross-pollinating with another application's ring. It is a required property.
- `channel` - An instance of TChannel. It is a required property.

- `hostPort` - The address of your Ringpop. This is used as the node's identity in the membership protocol and the ring. It is a required property.

NOTE: There are many other options one can specify in the Ringpop constructor. They are not yet documented.

3.2.2 `bootstrap(bootstrapFileOrList, callback)`

Bootstraps Ringpop; joins the cluster.

- `bootstrapFileOrList` - The path of a bootstrap file on disk. Its contents are expected to be a JSON array of Ringpop addresses. Ringpop will select a number of random nodes from this list to which join requests will be sent. Alternatively, this argument can be a Javascript array of the same addresses.
- `callback(err)` - A callback.

3.2.3 `handleOrProxy(key, req, res, opts)`

Acts as a convenience for the "handle or forward" pattern.

- `key` - An arbitrary key, typically a UUID. Its hash code is computed and its position along the ring is found. Its owner is the closest node whose hash code is closest (in a clock-wise direction)
- `req` - Takes the shape of a Node.js http.ClientRequest
- `res` - Takes the shape of a Node.js http.ServerResponse
- `opts` - Valid options are listed below.

opts

- `bodyLimit` - The maximum size of the allowable request body. Default is 1MB.
- `endpoint` - The TChannel endpoint to which the request will be forwarded. The default is /proxy/req. Typically, this should be left untouched.
- `maxRetries` - Maximum number of retries to attempt in the event that a forwarded request fails.
- `retrySchedule` - An array of delay multiples applied in between each retry. Default is `[0, 1, 3.5]`. These multiples are applied against a 1000ms delay.
- `skipLookupOnRetry` - A boolean flag to specify whether a request should be rerouted if the ring changes in between retries.
- `timeout` - The amount of time, in milliseconds, to allow for the forwarded request to complete.

3.2.4 `lookup(key)`

Looks up a key against the ring; returns the address of the Ringpop that owns the key.

- `key` - See the description of `key` above.

3.2.5 `lookupN(key, n)`

Looks up a key against the ring; returns the addresses of `n` distinct Ringpop's that own the key; useful for replication purposes. If `n` are not found, fewer addresses may be returned.

- `key` - See the description of `key` above.

- `n` - The number of secondary owners aka a "preference list".

3.2.6 `whoami()`

Returns the address of Ringpop

3.2.7 Events

Content coming soon...

3.3 An Example Express App

Let's see all of this come together in an example web application that you can run and curl yourself. This is a 3-node Ringpop cluster each with its own HTTP front-end capable of 'handling' and forwarding HTTP requests in less than 100 lines of code.

To run:

1. Paste this code into a file named `example.js`
2. `npm install ringpop@10.8.0 tchannel@2.8.0 express`
3. `node example.js`
4. curl to your heart's content: `curl 127.0.0.1:6000/objects/abc`

Node.js

```
var express = require(`express');
var Ringpop = require(`ringpop');
var TChannel = require(`TChannel');

var host = `127.0.0.1'; // not recommended for production
var ports = [3000, 3001, 3002];
var bootstrapNodes = [`127.0.0.1:3000', `127.0.0.1:3001',
    `127.0.0.1:3002'];

var cluster = [];

// Create Ringpop instances
ports.forEach(function each(port) {
    var tchannel = new TChannel();
    var subChannel = tchannel.makeSubChannel({
        serviceName: `ringpop'
    });

    cluster.push(new Ringpop({
        app: `yourapp',
        hostPort: host + `:' + port,
        channel: subChannel
    }));
});

// Bootstrap cluster
```

```
cluster.forEach(function each(ringpop, index) {
    ringpop.setupChannel();
    ringpop.channel.listen(ports[index], host, function onListen() {
        console.log(`TChannel is listening on ` + ports[index]);
        ringpop.bootstrap(bootstrapNodes,
            bootstrapCallback(ringpop, index));

        // This is how you wire up a handler for forwarded requests
        ringpop.on(`request', forwardedCallback());
    });
});

// After successfully bootstrapping, create the HTTP server.
var bootstrapsLeft = bootstrapNodes.length;
function bootstrapCallback(ringpop, i) {
    return function onBootstrap(err) {
        if (err) {
            console.log(`Error: Could not bootstrap ` + ringpop.whoami());
            process.exit(1);
        }

        console.log(`Ringpop ` + ringpop.whoami() + ` has bootstrapped!');
        bootstrapsLeft--;

        if (bootstrapsLeft === 0) {
            console.log(`Ringpop cluster is ready!');
            createHttpServers();
        }
    };
}

// In this example, forwarded requests are immediately ended. Fill in with
// your own application logic.
function forwardedCallback() {
    return function onRequest(req, res) {
        res.end();
    }
}

// These HTTP servers will act as the front-end
// for the Ringpop cluster.
function createHttpServers() {
    cluster.forEach(function each(ringpop, index) {
        var http = express();

        // Define a single HTTP endpoint that `handles' or forwards
        http.get(`/objects/:id', function onReq(req, res) {
            var key = req.params.id;
            if (ringpop.handleOrProxy(key, req, res)) {
                console.log(`Ringpop ` + ringpop.whoami() + ` handled ` + key);
                res.end();
            } else {
                console.log(`Ringpop ` + ringpop.whoami() +
                    ` forwarded ` + key);
            }
        });

        var port = ports[index] * 2; // HTTP will need its own port
        http.listen(port, function onListen() {
            console.log(`HTTP is listening on ` + port);
        });
    });
}
```

CHAPTER 4

Architecture, Design, and Implementation

4.1 Concepts

4.1.1 Membership Protocol

Ringpop implements a membership protocol that allows nodes to discover one another, disseminate information quickly, and maintain a consistent view across nodes within your application cluster. Ringpop uses a variation of the gossip protocol known as SWIM (Scalable Weakly-consistent Infection-style Process Group Membership Protocol) to disseminate membership updates across the many members of the membership list. Changes within the cluster are detected and disseminated over this protocol to all other nodes.

Ringpop uses the SWIM gossip protocol mechanisms of "ping" and "ping-req". Pings are used for disseminating information and fault detection. Members ping each other in random fashion until they get through the full membership list, rotate the list, then repeat the full round of pinging.

####SWIM Gossip Protocol for Information Dissemination Let's say you have a cluster with two nodes: A and B. A is pinging B and B is pinging A. Then a third node, C, joins the cluster after pinging B. At this point B knows about C, but A does not. The next time B pings A, it will disseminate the knowledge that C is now part of the cluster. This is the information dissemination aspect of the SWIM gossip protocol. ####SWIM Gossip Protocol for Fault Detection Ringpop gossips over TCP for its forwarding mechanism. Nodes within the ring/membership list are gossiping and forwarding requests over the same channels. For fault detection, Ringpop computes membership and ring checksums.

A membership list contains the addresses and statuses (alive, suspect, faulty, etc.) of the instances. It also contains additional metadata like the incarnation number, which is the logical clock. All this information is combined and we compute a checksum from it.

The checksums detect a divergence in the cluster in the event a request is forwarded, or a ping occurs, and the source and destinations checksums differ.

Ringpop retains members that are "down" in its membership list. SWIM manages membership status by removing down members from the list, whereas Ringpop keeps down members in the list allowing the ability to merge a split-brain after a network partition. For example, let's say two clusters form your application. If there isn't a way to identify which nodes were previously faulty or down because the network partition happened during that time, there would be no way to merge them back together.

4.1.2 Consistent Hashing

Ringpop leverages consistent hashing to minimize the number of keys to rebalance when your application cluster is resized. Consistent hashing in Ringpop allows the nodes to rebalance themselves with traffic evenly distributed. Ringpop uses FarmHash as its hashing function because it's fast and provides good distribution. Consistent hashing applies a hash function to not only the identity of your data, but also the nodes within your cluster that are operating on that data. Ringpop uses a red-black tree to implement its underlying data structure for its ring which provides log n, lookups, inserts, and removals.

Ringpop maintains a consistent hash ring of its members. Once members are discovered to join or leave the cluster, that information is added into the consistent hash ring. Then the addresses of the instances in the ring are hashed.

Ringpop adds a uniform number of replica points per node. To spread the nodes around the ring for a more even distribution, replica points are added for every node within the ring. It also adds a uniform number of replica points so the nodes and the hosts running these nodes are treated as homogeneous.

4.1.3 Forwarding

Ringpop offers proxying as a convenience and can be used to route your application's requests. Traffic through your application is probably directed toward a particular entity in your system like an object with an id. That id belongs somewhere in your cluster on a particular instance, depending on how it hashes. If the key hashes to an instance that did not receive the request, then that request is simply forwarded and everything is taken care of under the hood. This acts like a middleware layer for applications. Before the request even gets to your business logic, it is already routed to the appropriate node.

Ringpop has codified a handle or forward pattern. If a key arriving at instance A hashes to the node, it can process it, otherwise, it forwards it. This information is forwarded using a protocol called TChannel. TChannel is a networking framing protocol developed by Uber, used for general RPC. Ringpop uses TChannel as its proxying channel and transport of choice. It supports out-of-order responses at extremely high performance with benchmarks ranging from 20,000 to 40,000 operations per second.

Ringpop packs forwarded requests as HTTP over TChannel. HTTP is packed into the message that's transported over TChannel when it's forwarded, and then reconstructed on the other side.

Forwarding Requests

As an example, let's say node C joins a ring and now all of the addresses and replica points are evenly distributed around the ring. A, B, and C are pinging one another. The handle or forward pattern peforms a `ringpop.lookup`, gives it the sharding key and gets a destination back. If the destination resolves to A, then A can handle the request; otherwise it forwards it over TChannel transport to its destination.

Note: Eventually, this process will be moved to a Thrift model instead of HTTP.

4.2 How Ringpop Works

4.2.1 Joining a Cluster

1. The first node, A, checks a bootstrap list and finds that no other nodes are running.
2. Next, B starts up and has A to join. B reads the file from disk, then selects a random number of members. It will find A and start to form a consistent hash ring in the background, running within memory in Ringpop.
3. The nodes are positioned along the ring and exchange information with one another, forming a two-node cluster and pinging each other back and forth.

4.2.2 Handle or Forward

Upon arrival of a proxied request at its destination, membership checksums of the sender and receiver will be compared. The request will be refused if checksums differ. Mismatches are expected when nodes are entering or exiting the cluster due to deploys, added/removed capacity, or failures. The cluster will eventually converge on one membership checksum, therefore refused requests are best handled by retrying them.

Ringpop's request proxy has retries built in and can be tuned using two parameters provided at the time Ringpop is instantiated: `requestProxyMaxRetries` and `requestProxyRetrySchedule` or per-request with: `maxRetries` and `retrySchedule`. The first parameter is an integer representing the number of times a particular request is retried. The second parameter is an array of integer or floating point values representing the delay in-between consecutive retries.

Ringpop has codified the aforementioned routing pattern in the `handleOrProxy` function:

- returns `true` when key hashes to the "current" node and `false` otherwise.
- returns `false` and results in the request being proxied to the correct destination. Its usage looks like this:

```
var opts = {
    maxRetries: 3,
    retrySchedule: [0, 0.5, 2]
};

if (ringpop.handleOrProxy(key, req, res, opts)) {
    // handle request
}
```

4.2.3 Node Statuses

Content coming soon...

4.2.4 Flap Damping

Flap damping is a technique used to identify and evict bad nodes from a cluster. We detect flaps by storing membership update history and penalize nodes when flap is detected. When the penalty exceeds a specified suppress limit, the node is damped. When things go wrong and nodes are removed from the hash ring, you may see a lot of shaky lookups.

As an example, let's say A pings B, and B responds. Then, in the next round of the protocol, A pings B again but this time B is down. Then in the next round, A pings B, but this time B is up again. If there's a bad actor (a slow node that's overwhelmed by traffic), it's going to act erratically. So we want to evict it from the cluster as quickly as possible. The pattern of deviations between alive and suspect/faulty are known as flaps.

We detect flaps by storing the disseminated membership updates as part of the SWIM gossip protocol. When we detect a flap, we penalize the bad actor. Every node stores a penalty for every other node in the cluster. For example, A's view of B is different than C's view of B. When the penalty exceeds a certain suppression limit, that node is damped. That damped status is disseminated throughout the cluster and removed from the ring. It is evicted and penalized so that it cannot join the ring for a specified period of time.

If the damp score goes down and then decays, the problem is fixed and it will not be penalized and evicted from that ring. But if excessive flap exceeds the red line (damping threshold), then a damping sub-protocol is enacted similar to the indirect pinging sub-protocol defined by SWIM.

Say the damp score for B exceeds the red line. A fans out a damp-req request to *k* random members and asks for their damp score of B. If they also communicate that B is flapping, then B is considered damped due to excessive flapping. A marks B as damped, and disseminates that information using the gossip protocol.

4.2.5 Full Syncing

Content coming soon...

4.2.6 TChannel

Content coming soon...

TChannel is a network multiplexing and framing protocol for RPC. Some of the characteristics of TChannel:

- Easy to implement in multiple languages, especially JavaScript and Python.
- High performance forwarding path. Intermediaries can make a forwarding decision quickly.
- Request/response model with out-of-order responses. Slow request will not block subsequent faster requests at head of line.
- Large requests/responses may/must be broken into fragments to be sent progressively.
- Optional checksums.
- Can be used to transport multiple protocols between endpoints, e.g., HTTP + JSON and Thrift

Components

- tchannel-node: TChannel peer library for Node.js.
- tchannel-python: TChannel peer library for Python.
- tchannel-golang: TChannel peer library for Go.
- tcap: TChannel packet capture tool, for eavesdropping and inspecting TChannel traffic.
- bufrw: Node.js buffer structured reading and writing library, used for TChannel and Thrift.
- thriftrw: Node.js Thrift buffer reader and writer.
- thriftify: Node.js Thrift object serializer and deserializer with run-time Thrift IDL compiler.

4.3 Extensions

Ringpop is highly extensible and makes possible for a multitude of extensions and tooling built around it. Here are the libraries that extend Ringpop.

4.3.1 Sharding

Content coming soon...

4.3.2 Actor Model

Every actor in the system has a home (a node in the cluster). That node receives concurrent requests for every actor. For every actor, there is a mailbox. Requests get pulled off the mailbox one by one. Processing a request may result in new requests being sent or new actors being created. Each request that's processed one by one may result in some other request to another service, or a request for more actors to be spun up.

4.3.3 Replication

Building Redundancy with Ringpop.

Reliable Background Operations

Content coming soon...

Leader Election

Content coming soon...

CHAPTER 5

References

Learn more about key concepts related to Ringpop.

5.1 FAQ

5.2 Glossary

5.2.1 A

- **Actor model**: Concurrent computation model used by Ringpop that allows messages to arrive concurrently, then processed one by one. Messages are placed in a mailbox based on the sharding key, and then processed one by one. Each request that's processed one by one may result in some other request to another service, or a request for more actors to be spun up. Alive: A membership status signifying the node is healthy, and not suspect, faulty, or damped.

5.2.2 B

- **Bad actor**: A slow node that's overwhelmed by traffic.

5.2.3 C

5.2.4 D

- **Damped**: Flap damping is a technique used to identify and evict bad nodes from a cluster. Flaps are detected by storing membership update history and penalize nodes when flap is detected. When the penalty exceeds a specified suppress limit, the node is damped. The damped status is disseminated throughout the cluster and removed from the ring.

5.2.5 E

5.2.6 F

- **Flap damping**: Flap damping is a technique used to identify and evict bad nodes from a cluster.
- **FarmHash**: Hashing function used by Ringpop.

- **Faulty**: A state of the node that is reached after a defined “suspect” period, where a node is unstable or not responding to pings from other nodes. A suspect period will begin, and if it ends with the node not recovering, the node is considered faulty and is removed from the ring.

5.2.7 G

- **Gossip**: A type of protocol where nodes disseminate information about each other using pings.

5.2.8 H

- **Handle or forward**: This is Ringpop’s forwarding approach. If a key hashes to an instance that is not the one that received the request, then that request is simply forwarded to the proper instance and everything is taken care of under the hood. This acts like a middleware layer for applications that before the request even gets to your business logic, it is already routed to the appropriate node.
- **Hash ring**: Ringpop leverages consistent hashing to minimize the number of keys to rebalance when your application cluster is resized. Ringpop’s consistent hashing allows the nodes to rebalance themselves and evenly distribute traffic. Ringpop maintains a consistent hash ring of its members. Once members are discovered to join or leave the cluster, that information is added into the consistent hash ring. Then the instances’ addresses along that ring are hashed.

5.2.9 I

5.2.10 J

5.2.11 K

5.2.12 L

5.2.13 M

- **Membership list**: Ringpop uses a variation of SWIM to disseminate membership updates across the members of a membership list, which contains additional metadata like the incarnation number, instances’ addresses, and status (alive, suspect, faulty, etc.). Members ping each other in random fashion until they get through the full membership list, rotate the list, then repeat the full round of pinging.
- **Multi-cast**:

5.2.14 N

5.2.15 O

5.2.16 P

- **Ping**: Ringpop uses pings to disseminate information and for fault detection. Members ping each other in random fashion until they get through the full membership list, rotate the list, then repeat the full round of pinging.

5.2.17 Q

5.2.18 R

- **Replica points**: Ringpop adds a uniform number of replica points per node to spread the nodes around the ring for a more even distribution. Ringpop also adds a uniform number of replica points so the nodes and the hosts running these nodes are treated as homogeneous.
- **Ringpop**: Ringpop is a library that brings application-layer sharding to your services, partitioning data in a way that's reliable, scalable and fault tolerant.
- **Ringpop forwarding**:

5.2.19 S

- **SERF**: Gossip-based membership that exchanges messages to quickly and efficiently communicate with nodes.
- **Sharding**: A way of partitioning data, which Ringpop does at the application layer of your services in a way that's reliable, scalable and fault tolerant.
- **Suspect**: A state of the node where it is unstable or not responding to pings from other nodes. If nodes stay suspect during the pre-defined suspect period without recovering, it will then be considered faulty and removed from the ring.
- **SWIM**: Scalable Weakly-consistent Infection-style Process Group Membership Protocol

5.2.20 T

- **TChannel**: TChannel is a network multiplexing and framing protocol for RPC. TChannel is the transport of choice for Ringpop's proxying channel.

5.2.21 V

5.2.22 W

5.2.23 X

5.2.24 Y

5.2.25 Z

5.3 Use Cases

5.4 Papers

- BGP Route Flap Damping
- Dynamo: Amazon's Highly Available Key-value Store
- Efficient Reconciliation and Flow Control for Anti-Entropy Protocols
- Epidemic Broadcast Trees

- FarmHash
- Riak
- SWIM Presentation Slides by Armon Dadgar from Hashicorp
- SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol
- TChannel
- The Accrual Failure Detector
- Time, Clocks, and the Ordering of Events in a Distributed System

5.5 Presentations

CHAPTER 6

Community

6.1 Google Group

6.2 Contributing

6.3 License

Ringpop is available under the MIT license. See the LICENSE file for more info.