Improved Error Handling in Go API

Proposal: 0005
Authors: Operator Foundation
Status: Initial proposal

Implementation: TBD

Introduction

Transport modules must provide constructor functions. A module can provide either just a client factory, just a server factory, or both. These functions take transport-specific parameters, so there is no fixed interface definition. The functions below are examples for what these function signatures might look like for a hypothetical transport. These examples are taken from the PT 2.1 specification.

```
// Create outgoing transport connection
// The Dial method implements the Client Factory abstract interface.
Dial(address string) net.Conn

// Create listener for incoming transport connection
// The Listen method implements the Server Factory abstract interface.
Listen(address string) net.Listener
```

While the parameters taken by the factory functions can vary between transports, this proposal is for mandating that the return types of the factory functions include optional errors.

Motivation

Having Dial() and Listen() return optional errors is closer to how these functions are implemented in the Go standard library “net” package. Additionally, not having them return errors has been the source of bugs in the dispatcher application and not having errors available when Dial() and Listen() fail has made debugging more difficult.

Proposed solution

The Dial() and Listen() functions will be required to have return types that include optional errors, rather than just the values they currently return. This will allow more informative failures from Dial() and Listen(), resulting in easier debugging.

Design

Transport modules must provide constructor functions. A module can provide either just a client factory, just a server factory, or both. These functions take transport-specific parameters, so there is no fixed interface definition. The functions below are examples for what these function signatures might look like for a hypothetical transport. These examples are taken from the PT 2.1 specification.

```
// Create outgoing transport connection
// The Dial method implements the Client Factory abstract interface.
Dial(address string) net.Conn

// Create listener for incoming transport connection
// The Listen method implements the Server Factory abstract interface.
Listen(address string) net.Listener
```

While the parameters taken by the factory functions can vary between transports, this proposal is for mandating that the return types of the factory functions include optional errors. So the new API would look like this:

```
// Create outgoing transport connection
// The Dial method implements the Client Factory abstract interface.
Dial(address string) (net.Conn, error)

// Create listener for incoming transport connection
// The Listen method implements the Server Factory abstract interface.
Listen(address string) (net.Listener, error)
```

Effect on API Compatibility

This is a breaking change to the Go API, requiring a new major version number. It is considered a breaking change because the Go API functions Dial and Listen now have additional requirements not present in the previous revision of the specification. As no requirement on return types previously existed, existing implementations may or may not already be in compliance with the new API, depending on the choices made by the developers.

Effect on IPC Compatibility

This change only affects the Go API. There is no effect on IPC compatibility. The IPC implementation that uses the Go API internally will need to be modified to use the new Go API.

Alternatives considered

In the current reference implementation of the dispatcher, the Go API is used that does not incorporate the return values with errors. Instead, the returned net.Conn or net.Listener is checked to see if it is nil, indicating an error. This has several drawbacks. First, because it is not idiomatic in Go, it has led to several bugs where the implementation did not check whether the returned value was nil. In Go, this is a serious bug that can lead to the application crashing. Second, checking for a nil value indicates failure, but it does not indicate the source of failure. Returning an optional error is more informative for debugging.