Bridge Manager

Motivation:

The main event loop in zedrouter service is currently overloaded with several tasks including network instance handling, app network handling, network instance probing, network flow logging export, DNS/DHCP monitoring and metrics export. Several of these tasks require execution of external commands like dnsmasq, iptables, radvd etc. There were instances where the execution of these external commands took longer than normal, causing watchdog timeout reboots.

The number of access control rules used per application instance may be in single digits today, but the number can grow considerably with increased eve adoption and deployment in production environments. With external iptables, ip6tables commands being used for setting up acls and other external commands (dnsmasq, radvd etc) for application network setup, it is possible that slower (slow cpu, less memory) edge nodes spend considerable amounts of time executing these external command. The current code handles configuration of all network instances and application networks in the main event loop of zedrouter. Main event loop in zedrouter spending too much time configuring network instances and application networks can make watchdog modules to consider zedrouter as stuck and subsequently reboot edge nodes. Also, if zedrouter's main event loop is stuck (seen in some cases with iptables commands), any subsequent events/configuration updates also get blocked.

If we move the application networking setup code into a separate go-routine per network instance (call it bridge manager),

1. zedrouter's main event loop will be free for handling other non-configuration related events.
2. Issues corresponding to one network instance will not directly affect events corresponding to other network instances.
3. With network instance management segregated, it will be easier to enforce any limits on the resources used per network instance.

Bridge manager:

The idea is to create a subtask(go-routine) per network instance in zedrouter. This new sub task will be called bridge manager. Bridge manager will be a separate go-routine running parallel to the main event loop inside zedrouter and is responsible for managing a given network instance and the application networks attached to the corresponding network instances. There will be as many bridge managers as the number of network instances configured on device.

When zedrouter receives configuration for a new network instance, it creates a new bridge manager go-routine that takes care of network instance specific setup. Zedrouter will start the new bridge manager as a go-routine with network instance name as the only argument. Bridge manager during its initial startup subscribes to NetworkInstanceConfig updates from pubsub. It also gets the latest network instance configuration from pubsub to finish it's initial setup. Further updates to network instance configuration will be directly received by the bridge manager via pubsub. Bridge manager on receiving delete for its NetworkInstanceConfig object will clean up any created state/system configuration and exit.

Bridge manager takes care of creating the bridge interfaces, setting up network instance type specific services (eg: strongswan service setup for VPN type), setting up initial dnsmasq configuration and starting dnsmasq (if required) corresponding to the network instance. After the initial setup, the bridge manager enters an event loop where it listens for further configuration updates for the network instance and application adapter network configuration for application networks.

Bridge manager also takes care of any network instance - type specific setup when the network instance is activated. For ex: Creating the masquerade rules, network instance specific route tables and PBR rules for NAT network instances, strongswan setup for VPN network instances etc. Corresponding network instance status will also be published by the associated bridge manager only.

Bridge manager will also open netlink socket to get updates about route changes and link changes. Network instance specific route tables will be updated as required.

Currently, uplink probing happens as part of the main event loop in zedrouter and is a common loop for all network instances. We can move this probing out of the main event loop and then fold per-network instance probing logic into the bridge manager event loop. Also, every bridge manager will subscribe to device network status separately for responding to changes in uplink ports and changes the probing database and network instance attached ports as required. This has the advantage of splitting the existing giant probing loop into per network instance bridge manager loops. Given that the initial design for probing is done with future possibility to specify probe targets and methods - ping vs http get etc, having uplink port probing per network instance becomes mandatory.

DNS, Dhcp monitoring can also be folded into the same bridge manager event loop.

Application network:

Zedrouter's main event loop will create a separate agent scope (UUID of network instance as scope) publishers (to which individual bridge managers subscribe) per bridge manager (network instance) and publish the application adapter configuration (AppAdapterConfig). This new AppAdapterConfig object will have a key with combination of application UUID and the offset index of underlay inside application network configuration. Bridge manager allocates any required resources for the application adapter (ip address etc) along with applying any application specific acls. Bridge manager then publishes application adapter network status (AppAdapterStatus). AppAdapterConfig object along with the AppAdapterStatus object will also carry a TargetVersion field (monotonically increasing number starting from 1) that helps in the reconciliation process of zedrouter with bridge managers. Similarly, the AppNetworkStatus structure will also be enhanced to have this new TargetVersion field for comparing with incoming AppAdapterStatus messages from individual bridge managers. Zedrouter's main event loop subscribes to AppAdapterStatus trying to reconcile application network configuration with available application adapter network status. AppAdapterStatus will be very similar to existing UnderlayNetworkStatus object with some additional fields to make a unique key per application per underlay. Once the main event loop receives status for all application adapters, the completed application network status will be published. Also, since the AppAdapterConfig objects are published to bridge managers using pubsub, zedrouter main event loop will not need an explicit confirmation of bridge manager status.

type AppAdapterConfig struct {

// Key = AppUUID + UnderlayIndex

AppUUID string

UnderlayIndex int

TargetVersion int

UnderlayConfig UnderlayNetworkConfig

}

type AppAdapterStatus struct {

AppUUID string

UnderlayIndex int

TargetVersion int

Activated bool

UnderlayStatus UnderlayNetworkStatus

}

Main event loop in zedrouter in the events of application network add/active and deactivate/delete will wait for an update to AppAdapterStatus from all required bridge managers before publishing the final AppNetworkStatus. This is to make sure that the other agents down the path do not change application instances prematurely (eg: domainmgr).

The Pending flag in AppNetworkStatus will be cleared only after the following in AppNetworkConfig and AppAdapterStatus match.

1. Activate state in configuration should match the Activated flag in all AppAdapterStatus messages.
2. TargetVersion should match between each AppAdapterConfig and the corresponding AppAdapterStatus message.
3. TargetVersion should match between AppNetworkStatus and it's AppAdapterStatus updates.

In case if the Activate flag of one or more AppAdapterStatus messages do not match the configuration - while the TargetVersion is matching, zedrouter will send a new AppAdapterConfig towards the respective bridge manager with the same TargetVersion. This is to make sure that individual bridge managers see the same copy of AppNetworkConfig as seen by the main event loop in zedrouter.

Message Exchanges:

Bridge manager Creation:

As discussed already, each network instance will have one bridge manager go-routine created that does the chores corresponding to the given network instance.

When the main go-routine in zedrouter receives a new network instance (previously not seen), it calls the bridge manager create function that does any required setup and launches a new go-routine. This new go-routine (dubbed bridge manager) finishes the creation of any bridge interfaces along with dnsmasq configuration and launches any required processes. Bridge manager then jumps into a select loop that watches for any AppAdapterConfig objects coming from the main event loop for creation of app networks. Bridge manager also publishes AppAdapterStatus after successful initialization. Main zedrouter event loop subscribes to NetworkInstanceStatus from bridge managers in order to keep track of the currently running bridge managers by maintaining a map data structure of the currently running bridge managers.

onlineBridgeManagers map[uuid.UUID]NetworkInstanceStatus

App network Create/Modify:

AppNetworkConfig objects come with a list of UnderlayNetworkConfig objects that describe it's network connections. Each of these application underlay networks will be connected to individual network instances that are implemented by specific bridge manager go-rotines.

Main event loop in zedrouter publishes AppAdapterConfig to required bridge managers irrespective of them being online. Bridge managers that get started late shall receive all pending AppAdapterConfig objects by subscribing to the agent scope pubsub channel corresponding to their UUID. Any AppAdapterStatus objects with their corresponding bridge managers offline (not present) will be marked accordingly (with error message in status) in AppNetworkStatus and the entire AppNetworkStatus will be kept in pending state till all required network instances (bridge managers) get created.

Rapid configuration updates:

When the AppNetworkConfig corresponding to a particular app is still under process (not finished with setup), zedrouter may receive subsequent activate /deactivate, modify/delete messages for the same app network. Zedrouter to tackle this case will use a version based target counter in AppNetworkStatus (TargetVersion field in AppNetworkStatus). Every time a new update for an existing AppNetworkConfig arrives, zedrouter increments the TargetVersion field, moves AppNetworkStatus into pending state and sends off the required AppAdapterConfig objects to respective bridge managers. Only when all required bridge managers send back AppAdapterStatus with TargetVersion expected, will the zedrouter move AppNetworkStatus out of pending state. All intermediate AppAdapterStatus updates from individual bridge managers will be updated in AppNetworkStatus by zedrouter and published (still in pending state). This works in eventual consistency mode, where AppNetworkStatus objects transition through a sequence of changes before reaching a state that is consistent with their corresponding latest AppNetworkConfig from the cloud.

When bridge managers receive network instance delete messages with application networks still connected, the bridge manager will destroy all application adapters (including any state) connected to it and publish corresponding AppAdapterStatus messages. This way zedrouter knows that there are changes in AppNetworkStatus.