Meshnet gRPC-Wire Reconciliation Design

Authors: Alex Masi (Google)
Manomugdha Biswas (Keysight)
Kingshuk Mandal (Keysight)

Reviewers: Marcus Hines (Google)
Michael Kashin (Cilium)

Table of Contents

Background

Meshnet supports a plugin to use gRPC based wires to connect containers running across different nodes (add/or different hosts) in a k8 cluster. The meshnet daemon creates the “wires” and stores wire info + pcap handles in local memory: https://github.com/networkop/meshnet-cni/blob/d3ae64833f4710c0ba1810737df0bd5de414844f/daemon/grpcwire/grpcwire.go#L143

Meshnet stores general link information in the CRD Topology + Link types: https://github.com/networkop/meshnet-cni/blob/d3ae64833f4710c0ba1810737df0bd5de414844f/api/types/v1beta1/topology.go#L37

Meshnet daemon uses the metadata stored in the CRD to get link info per pod: https://github.com/networkop/meshnet-cni/blob/d3ae64833f4710c0ba1810737df0bd5de414844f/daemon/meshnet/handler.go#L31

Issue

Overview here: https://github.com/networkop/meshnet-cni/issues/57

The fact that grpc wire info is only stored in local memory means that meshnet pod restarts (due to crashes/reschedules) causes the wire info to be lost, and meshnet does not properly reconcile the wire info leading to grpc wires not functioning.

This is confirmed to be an issue during meshnet topology creation, unclear if this would be an issue if the topology was successfully created and then a meshnet pod crashed. Either way this is a large issue as the number of pods in topologies increases.

Proposal

Persist the grpc wire info (including corresponding pcap handles) in the meshnet topology CRD, so meshnet pod bring up can cleanup/reopen the pcap handles to reestablish grpc wire connectivity.

Current Behavior

Adapted from https://github.com/networkop/meshnet-cni/tree/d3ae64833f4710c0ba1810737df0bd5de414844f

1. Kubernetes cluster gets populated with the topology information via custom resources
2. pod-1/pod-2 come up, local kubelet calls the meshnet binary for each pod to setup their networking.
3. Based on the CNI configuration file, Kubelet calls meshnet to set up additional interfaces.
4. meshnet binary updates the topology data with pod's runtime metadata (namespace filepath and primary IP address).
5. meshnet binary (via a local meshnet daemon) retrieves the list of links and looks up peer pod's metadata to identify what kind of link to setup - veth, grpc or macvlan.
6. If the peer is on the same node, it calls koko to setup a veth link between the two pods.
7. If the peer is on the remote node, it makes a gRPC AddGRPCWireRemote/Local and stores the pcap handle + wire info in local memory

Proposed Behavior

1. Kubernetes cluster gets populated with the topology information via custom resources
2. pod-1/pod-2 come up, local kubelet calls the meshnet cni-plugin for each pod to setup their networking.
3. Based on the CNI configuration file, Kubelet calls meshnet to set up additional interfaces.
4. meshnet cni-plugin updates the topology data with pod's runtime metadata (namespace filepath and primary IP address).
5. meshnet cni-plugin (via a local meshnet daemon) retrieves the list of links and looks up peer pod's metadata to identify what kind of link to setup - veth, vxlan or macvlan.
6. If the peer is on the same node, it calls koko to setup a veth link between the two pods.
7. If the peer is on the remote node, it does two things:
    a. It makes a gRPC AddGRPCWireRemote/Local
        i. **AddWire writes to CRD status fields.**

On daemon reboot, the dynamic clientset is used to lookup the grpc wire info.

Metadata options

Create new CRD to store the wire status in K8S data store

```
// Collection of all the GRPC wires UID(s) for single Kubernetes node (KNode)
type GWireKNodeSpec struct {
    metav1.TypeMeta        `json:",inline"`
    UIDs             []int `json:"uids"`
}


// Collection of all the GRPC wires across the topology
type GWireKObjList struct {          <---- KObj => “Kubernetes Object”
    metav1.TypeMeta   `json:",inline"`
    metav1.ListMeta   `json:"metadata,omitempty"`
    Items []GWireKObj `json:"items"`  <---- details for every node in the topo
}


// Collection of all GRPC wires for a single node
type GWireKObj struct {
    metav1.TypeMeta         `json:",inline"`
    metav1.ObjectMeta       `json:"metadata,omitempty"` <-- has namespace to which
                                                             a topology belongs to
    Status GWireKNodeStatus `json:"status"` <--all GRPC wires for a single node
    Spec   GWireKNodeSpec   `json:"spec"`
```

```
}
type GWireKNodeStatus struct {
    metav1.TypeMeta           `json:",inline"`
    GWireKItems []GWireStatus `json:"gWireKItem"`
}

// Details of an individual wire
type GWireStatus struct {
    LocalNodeName            _string `json:"node_name"`
    LinkUid                  _int64  `json:"link_uid"`
    TopoNamespace            _string `json:"topo_namespace"`
    LocalPodNetNs            _string `json:"local_pod_net_ns"`
    LocalPodName             _string `json:"local_pod_name"`
    LocalPodIp               _string `json:"local_pod_ip"`
    LocalPodIfaceName        _string `json:"local_pod_iface_name"`
    WireIfaceNameOnLocalNode _string `json:"wire_iface_name_on_local_node"`
    WireIfaceIdOnPeerNode    _int64  `json:"wire_iface_id_on_peer_node"`
    GWirePeerNodeIp          _string `json:"gwire_peer_node_ip"`
}
```

When to create the GRPCWire custom resource.

Meshnet CNI follows Container Network Interface (CNI) Specification. Meshnet is composed of two components. A meshnet-CNI-plugin and meshnet-daemon set. The plugin interacts with Kubernetes system and the meshnet-daemon set runs as background component and works as per instruction sent by the meshnet-plugin.

As per CNI specification, container runtime invokes plugin binary as and when the runtime needs to provision the container network. CNI specification defines the protocol between the plugin binary and the runtime.

CNI specification defines 4 operations: ADD, DEL, CHECK, and VERSION. It expects the plugin implementer to support these 4. For this design doc we will focus on ADD and DEL.

Purpose of ADD is to add container to network, create interfaces and more depending on the layer at which the CNI is expected to work. Meshnet is a MAC layer CNI, whose objective is to provide L1 point to point connectivity (we will refer it as a wire) between two pods running in two different nodes. For meshnet, ADD call back is implemented by meshnet CNI plugin (we will refer it as `cmdAdd`) and is invoked by container runtime. In `cmdAdd`, meshnet CNI does the following: -

1. Creates the interface as per the meshnet Topology CR.
2. Assign randomly generated MAC address to interface.
3. Set the correct interface properties. For example – disable all HW acceleration flag for this virtual interface as this CNI does not provide any hardware acceleration.
4. If the mode is VxLAN then it set up the VNI number etc.
5. If the mode is gRPC, then it sets up the gRPC tunnels across Nodes

When our interest is to add reconciliation logic for gRPC (for now, will add VxLAN reconciliation later), then we need to do some additional work in step 5. A point to point “wire” has two ends, a local end and peer end. In the add call back, local end of wire is created by the function `CreateGRPCWireLocal` and the remote end of the wire is created by the function `CreateUpdateGRPCWireRemoteTriggered`. Both functions create the GRPC wire and store it “in memory” storage for meshnet daemon. In case for any reason the meshnet daemon is restarted then this “in memory” information is lost.

To reconcile, we will create the GRPCWire custom resource just after successful a grpc-wire in “in memory” storage. This CRD (given below) will store the necessary information in Kubernetes data-store. This persistent storage will enable the meshnet daemon to reconcile the existing Topology in case the daemon reboots.

GRPCWire CRD

```
Few lines in descriptions are deleted to fit them in this doc.
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    controller-gen.kubebuilder.io/version: v0.11.1
  creationTimestamp: null
  name: gwirekobjs.networkop.co.uk
spec:
  group: networkop.co.uk
  names:
    kind: GWireKObj
    listKind: GWireKObjList
    plural: gwirekobjs
    singular: gwirekobj
  scope: Namespaced
  versions:
  - name: v1beta1
    schema:
      openAPIV3Schema:
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values.
            type: string
          kind:
            description: 'Kind is a string value representing the REST resource this
              object represents. Servers may infer this from the endpoint the client
              submits requests to. Cannot be updated. In CamelCase.
            type: string
          metadata:
            type: object
          spec:
            properties:
              apiVersion:
                description: 'APIVersion defines the versioned schema of this representation
                  of an object. Servers should convert recognized schemas to the latest
                  internal value, and may reject unrecognized values.
                type: string
              kind:
```

```
            description: 'Kind is a string value representing the REST resource
              this object represents. Servers may infer this from the endpoint
              the client submits requests to. Cannot be updated. In CamelCase.
            type: string
          uids:
            description: unique link id
            items:
              type: integer
            type: array
        type: object
      status:
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of this representation
              of an object. Servers should convert recognized schemas to the latest
              internal value, and may reject unrecognized values.
            type: string
          grpcWireItems:
            items:
              properties:
                gwire_peer_node_ip:
                  description: peer node IP address
                  type: string
                link_id:
                  description: Unique link id as assigned by meshnet
                  format: int64
                  type: integer
                local_pod_iface_name:
                  description: Local pod interface name that is specified in topology
                    CR and is created by meshnet
                  type: string
                local_pod_ip:
                  description: Local pod ip as specified in topology CR
                  type: string
                local_pod_name:
                  description: Local pod name as specified in topology CR
                  type: string
                local_pod_net_ns:
                  description: Netwokr namespace of the local pod holding the
                    wire end
                  type: string
                node_name:
                  description: Name of the node holding the wire end
                  type: string
                topo_namespace:
```

```
                  description: The topology namespace.
                  type: string
                wire_iface_id_on_peer_node:
                  description: The interface id, in the peer node adn is connected
                    with remote pod. This is used for de-multiplexing received
                    packet from grpcwire
                  format: int64
                  type: integer
                wire_iface_name_on_local_node:
                  description: The interface(name) in the local node and is connected
                    with local pod
                  type: string
              type: object
            type: array
          kind:
            description: 'Kind is a string value representing the REST resource
              this object represents. Servers may infer this from the endpoint
              the client submits requests to. Cannot be updated. In CamelCase.
            type: string
        type: object
    type: object
served: true
storage: true
```

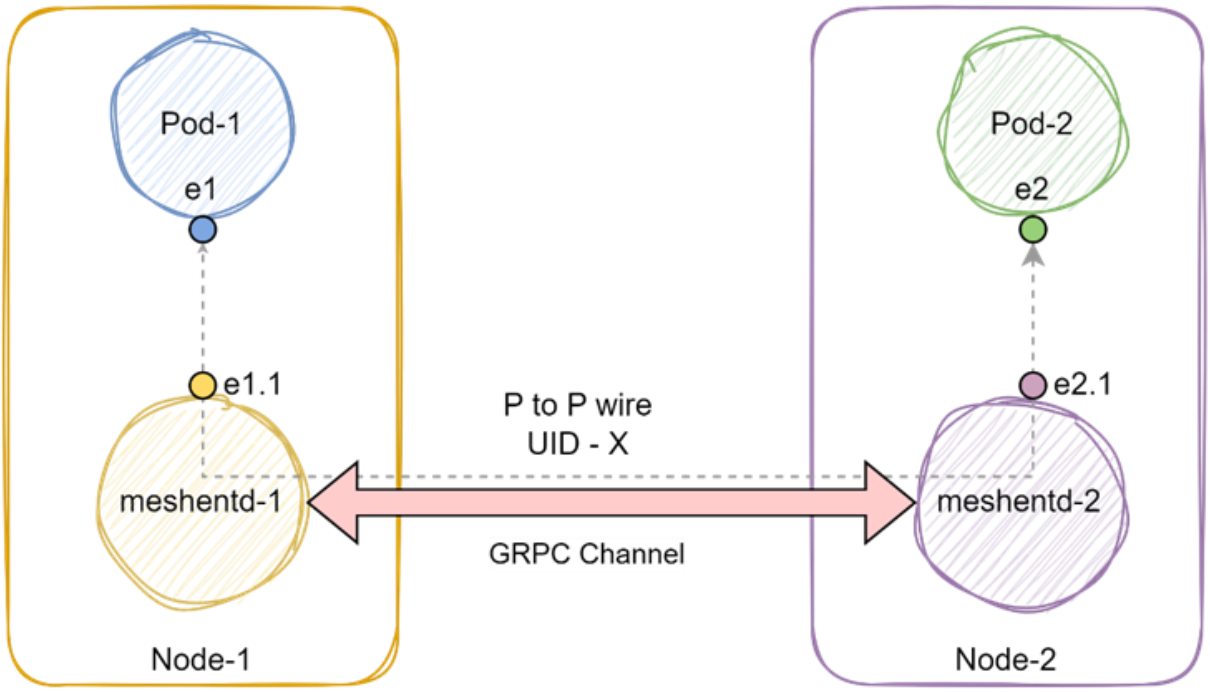
Purpose of DEL is to remove a container from the network and release any networking resources it was occupying, for example releasing IP address. A container can be removed when the end user wants to delete it. A container can also be removed from a cluster node when Kubernetes wants to relocate a pod from one node to another. The CNI protocol expects every CNI plugin to provide a DEL call-back (we will refer it as `cmdDel`) to handle this delete operation.

Meshnet-CNI will do the following to cmdDel call-back to clean-up `GRPCWire` CR:-

1. Remove the interface created for the pod, for which it is handling `cmdDel`
2. From K8s data store Clean up the GRPCWire CR for the local end of the wire.
3. Notify the remote node to do the GRPCWire CR clean up at the remote end.
4. The function `DeleteWiresByPod` will need to be updated to handle all the delete related actions.

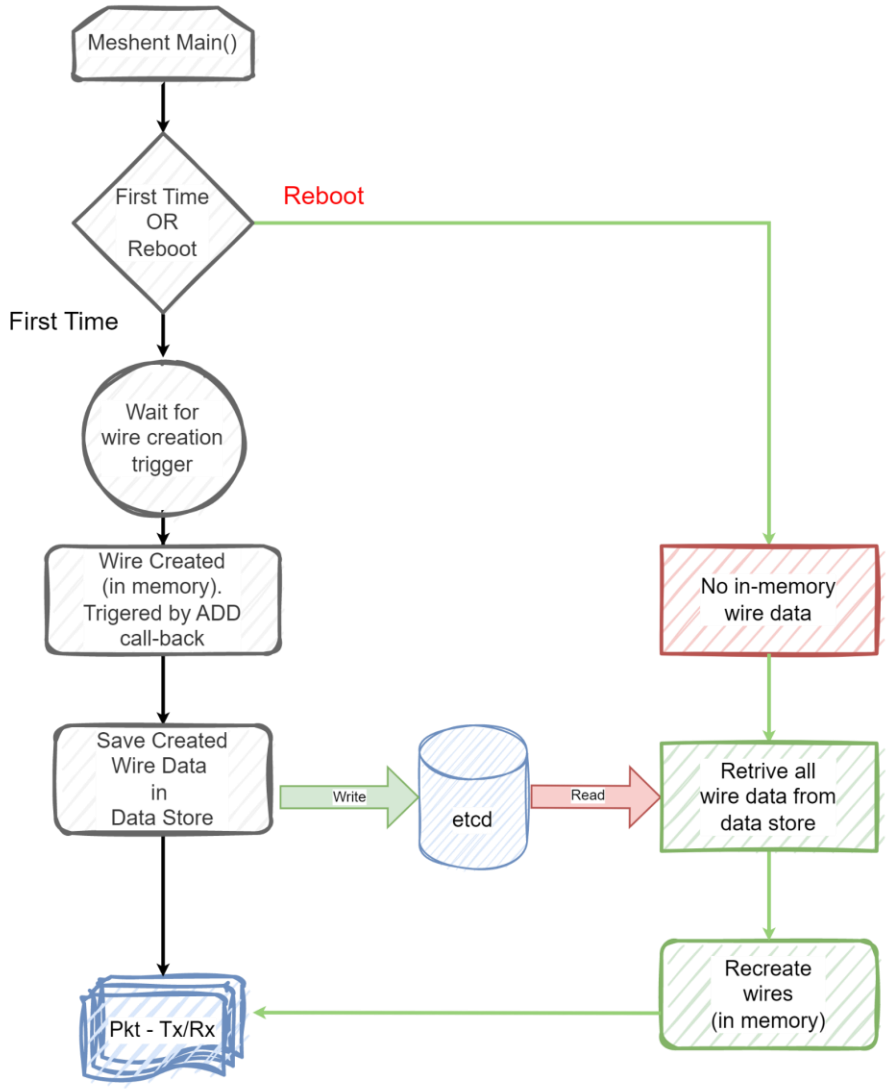
How the reconciliation will work

The objective of reconciliation is to bring up the pre-created topology on meshnet daemon reboot. The topology is created for the first time by Kubernetes runtime with the help of add-call-back as explained in earlier sections.



Referring to the picture above, the P-to-P grpc wire was created by executing add call-back. Post creation, the wire connectivity information and the created interfaces information are stored as part of GRPCWire CR. These are stored in persistent Kubernetes data-store etc. An Example of connectivity information is Wire `UID-X` is established between `Node-1` and `Node-2` etc.

The high-level reconciliation workflow:



If meshnet daemon reboots in any one of the nodes, then the following actions to be taken to reconcile the wire connectivity: -

Step 1 is for the daemon to detect that it has been rebooted and it is not the first time start. This is an important condition to detect. For the first time start, the wire creation will happen through the 'add function' but in subsequence reboots the reconciliation logic must bring up the wire. So, when the meshnet daemon starts, it checks with the K8S data store if there is any wire info specific to this node where this daemon has started/restarted. Say `meshnetd-1` does not find any wire info for `Node-1`. Then it's a fresh start for `meshnetd-1`. However, if it finds any wire info for `Node-1` then this is a reboot. For any given node, it's the meshent daemon that is running on that node writes the wire info (specific to that node) in the data store. So, when `meshnetd-1` finds info specific to `node-1`, that ensures that `meshnetd-1` has started before and hence this is a reboot.

Step 2 is to do the job of reconciliation. The add call back saves the following information in the K8S data store. A truncated version of the GRPCWire CRD is pasted below. It also shows the values (in

the context of the `node-1` in picture above) that were saved in the persistent storage. This information is enough to re-create (after ensuring that wires are not already existing) the `wire X` in `Node-1`. With this information `meshnetd-1` can re-store the in-memory data storage (`type wireMap struct`) that meshnet daemon uses for packet Tx/Rx over grpc wire.

```
// truncated from the CRD definition given earlier.
grpcWireItems:
    items:
      properties:
        namespace:                      <----- my topology namespace
        node_name:                      <---- node-1
        link_uid:                       <----- X
        wire_iface_name_on_local_node:  <----- e1.1
        local_pod_iface_name:           <----- e1
        local_pod_name:                 <----- pod-1
        wire_iface_id_on_peer_node:     <----- e2.1
        gwire_peer_node_ip              <----- ip address of node-2
```

With the above information the meshnet daemon can restart the Packet collection thread (`RecvFrmLocalPodThread`) to collect packet from `pod-1` and transport over GRPCWire to `node-2`. Recreate the necessary handles to deliver all the received packet (over with X) to `pod-1`