Mizar Bandwidth QoS for K8s Pods

# Motivation & Use Case

In order to better optimize network bandwidth usage in a system, one potential approach is to co-locate applications which use bandwidth at a steady rate but can tolerate burstiness with interactive applications that require certain bandwidth guarantees and greater priority.

For example, a file download process can co-exist with a chat-relay application if the lower bandwidth chat application can be prioritized higher than the file download process.

The idea is to enable applications that can handle transient periods of bandwidth paucity and be able to take advantage of capacity unused by latency-sensitive lower bandwidth applications.

# Goals & Non-goals

We plan to support the following:

1. Allow users to classify pods into two categories:
   1. High network priority - no bandwidth policing, priority transmission of egress traffic
   2. Low network priority - high network priority traffic usage leftover bandwidth available for use
2. Egress bandwidth QoS: Send high network priority traffic ahead of low network priority traffic
3. Place limits of caps on bandwidth available to low network priority traffic.
   1. This limit may increase or decrease based on average high network priority traffic bandwidth usage. (TBD: Determine if we really need to implement this)
4. Stretch: Measuring high network priority pod traffic bandwidth usage and adjust the aggregate bandwidth available to low network priority pods on a node.

The following are non-goals:

1. Supporting ingress bandwidth QoS or policing is a non-goal.
2. Multihomed systems (systems with multiple NICs) is not a goal for this release. (for 5/30)
3. Support for per-pod traffic shaping or rate limiting. (for 5/30 release)
   1. In this release, we focus on ensuring that the sum of bandwidth used by all low network priority pods does not exceed a defined aggregate rate limit.

# API Changes

Kubernetes Pod Spec allows users to specify bandwidth limits for pods via annotations.

For example:

|  |
| --- |
| **apiVersion**: v1  **kind**: Pod  **metadata**:  **annotations**:  **kubernetes.io/ingress-bandwidth**: 1M  **kubernetes.io/egress-bandwidth**: 1M  **...** |

We leverage these already available annotations to implicitly classify a pod’s network priority. If the annotation “kubernetes.io/egress-bandwidth” is specified, then the pod is considered a low network priority pod.

Cilium uses a [similar approach](https://docs.cilium.io/en/v1.9/gettingstarted/bandwidth-manager/) and leverages annotations to let users place bandwidth limits on pods.

By extending the meaning of egress-bandwidth annotation, no API changes are required to support allowing users to classify pods into high and low network priority pods.

## Co-Location of High and Low Network Priority Pods

We can continue to rely on the current scheduling mechanisms to evenly distribute high and low network priority pods based on the evaluation of priorities and predicates without needing any changes.

Hence, no scheduler changes are anticipated at this time. This may change if actual testing or real-world observations show that co-location factor is far from what is desired.

## Configuration Objects

When high network priority and low network priority pods are co-located, the user can recommend an initial aggregate bandwidth limit for low network priority pods.

We extend mizar-k8s-config configmap that is consumed by Mizar operator to hold a new parameter:

**aggregate-egress-bw-limit-percent**: Specifies the portion of available network bandwidth on a node to which low network priority traffic is capped. This limit may increase or decrease based on observed bandwidth use by high network priority pods. If 0, no aggregate limit is enforced.

# Design

This section details the following:

1. Handling of kubernetes.io/egress-bandwidth in Mizar
2. Implementation of network bandwidth priority and limits

## Pod’s kubernetes.io/egress-bandwith annotation handling

When a pod is scheduled, the node responsible for the pod sets up the pod network namespace and runs the containers specified in the pod. The Mizar operator is responsible for provisioning the eBPF map entry for the pod used by transit agent XDP. When egress-bandwidth annotation is present, eBPF map is configured with non-zero egress-bw value. [**TODO**: Determine the exact struct. Is it agent\_metadata\_t?? ]

**egress-bw**: New uint32 field that specifies the egress bandwidth limit for pod in Mbps. 0 means no egress bandwidth limit.

This enables the egress flow processing by Transit Agent to determine if a pod is subject to high or low priority network bandwidth and QoS policing.



The Mizar Operator and Endpoint CRDs are extended to read and set egress-bandwidth value.

## Egress Traffic Shaping

If egress-bw in the eBPF map bears a non-zero value, then packet egressing that pod is subject to network bandwidth rate limiting.

There is an aggregate rate limit enforced on network bandwidth available to low network priority pods. This limit is a percentage of the total available bandwidth of the NIC and is configured by the user via aggregate-egress-bw-limit-percent value in mizar-k8s-config configmap.

The Mizar operator reads this value and sends it to the transit daemons on each node in the cluster.

Note, there are two options to achieve egress traffic shaping:

* [EDT](https://netdevconf.info/0x14/pub/papers/55/0x14-paper55-talk-paper.pdf) (Earliest Departure Time) and eBPF
* Linux Traffic Control (tc)

### Earliest Departure Time

This method of rate-limiting of low network priority packets is implemented by the XDP program that runs in the droplet. Currently, this XDP only processes ingress packets.



The following changes are made:

* Mizar transit daemon configures the eBPF map [**TODO**: Identify suitable node-global struct to hold this value] that is accessed by the Transit XDP program on the droplet.
* Transit Agent XDP program on veth egress sets the last Reserved bit in the GENEVE header for pods with non-zero egress-bw, indicating that it is a low network priority packet.
* Transit XDP program on eth0 is extended to process egress packets sent to it. During the handling of egress traffic, it checks the last Reserved bit in GENEVE header, and if set, it subjects the packet to EDT algorithm. This bit is cleared before transmitting the packet.

Note: With this approach, we can easily add per-pod bandwidth policing and shaping of low network priority pod traffic in Transit Agent XDP upon egress from the veth pair.

### Linux Traffic Control

This approach intends to leverage the host networking stack TC functionality instead of XDP on droplet Tx. Low network priority egress packets are sent to a aggregate rate limited TBF instead of XDP\_REDIRECT to egress droplet interface. [TODO: Investigate if this is even possible.]

In this approach, transit daemon sets up the TBF vs programming EDT bw-limit-pct in the above approach.

## Aggregate Bandwidth Monitoring and Dynamic Tuning

The bandwidth used by low network priority pods is grouped together into a single aggregate, and a variable limit is placed on the network bandwidth they can use as a group. This limit is increased or decreased based on the average bandwidth used by the high network priority pods i.e all pods not annotated with egress-bandwidth. This requires bandwidth usage monitoring in order to determine and recommend a value for the aggregate limit.



TODO: Many questions need answers, more investigation needed. Add lots more details…

* What’s the best way to do this? A daemonset that runs a bw usage monitoring tool?
* Investigate what tool works best.
* Are there per-pod stats available? Or do we need to add stats gathering at XDP?

## Multihomed Hosts

If a host has more than one NIC, the above design would continue to work because pod egress traffic redirect to NIC is deterministic. This needs a closer look though to see if co-located pods are evenly distributed with egress interface assignments.