Pod network duplication in chaos engineering refers to intentionally creating duplicate network traffic or packets within a Kubernetes cluster to simulate network-related faults and test the resilience of the system. This technique is used to evaluate how well the system handles scenarios where network packets are duplicated, leading to potential issues such as packet loss, out-of-order delivery, or unexpected behavior in applications.

Here's how pod network duplication can be implemented in chaos engineering experiments:

1. \*\*Packet Duplication\*\*: Introduce faults in the network infrastructure or network overlay used in the Kubernetes cluster to duplicate network packets. This can be achieved through network proxies, middleboxes, or by modifying network configurations to replicate packets.

2. \*\*Traffic Injection\*\*: Inject duplicate network traffic at specific points in the network topology, such as between pods, nodes, or clusters. This simulates scenarios where network packets are duplicated due to misconfigurations, network congestion, or faulty hardware.

3. \*\*Randomization\*\*: Duplicate network traffic randomly or based on predefined criteria during chaos engineering experiments to simulate realistic scenarios. This helps uncover potential issues related to duplicate packets that may arise under varying conditions and workload patterns.

4. \*\*Observation and Analysis\*\*: Monitor the behavior of applications, services, and the underlying infrastructure during the network duplication experiment. Collect metrics related to packet loss, latency, throughput, and application performance to assess the impact of duplicated network traffic on the system.

5. \*\*Validation\*\*: Validate the system's resilience and fault tolerance by analyzing how well it handles duplicate network packets. Evaluate whether applications gracefully degrade, recover, or fail over in response to duplicated network traffic, and identify any areas for improvement in the network infrastructure or application design.

By intentionally introducing pod network duplication in chaos engineering experiments, organizations can proactively identify and mitigate potential issues related to network redundancy, packet duplication, and network reliability in Kubernetes clusters, thereby improving the overall resilience and robustness of their systems.

**Impact**

The impact of Pod Network Duplication chaos attack can be seen using: **ping <pod-ip> , through another cluster pod** command in the desired namespace.

|  |  |  |
| --- | --- | --- |
| **Variables** | **Description** | **Notes** |
| NETWORK\_INTERFACE | Name of ethernet interface considered for shaping traffic |  |
| TARGET\_CONTAINER | Name of container which is subjected to network latency | Optional | Applicable for containerd & CRI-O runtime only. Even with these runtimes, if the value is not provided, it injects chaos on the first container of the pod |
| NETWORK\_PACKET\_DUPLICATION\_PERCENTAGE | The packet duplication in percentage | Optional | Default to 100 percentage |
| CONTAINER\_RUNTIME | container runtime interface for the cluster | Defaults to containerd, supported values: docker, containerd and crio for litmus and only docker for pumba LIB |  |
| SOCKET\_PATH | Path of the containerd/crio/docker socket file | Defaults to /run/containerd/containerd.sock |  |
| TOTAL\_CHAOS\_DURATION | The time duration for chaos insertion (seconds) | Default (60s) |  |
| TARGET\_PODS | Comma separated list of application pod name subjected to pod network corruption chaos | If not provided, it will select target pods randomly based on provided appLabels |  |
| DESTINATION\_IPS | IP addresses of the services or pods or the CIDR blocks(range of IPs), the accessibility to which is impacted | comma separated IP(S) or CIDR(S) can be provided. if not provided, it will induce network chaos for all ips/destinations |  |
| DESTINATION\_HOSTS | DNS Names/FQDN names of the services, the accessibility to which, is impacted | if not provided, it will induce network chaos for all ips/destinations or DESTINATION\_IPS if already defined |  |
| SOURCE\_PORTS | ports of the target application, the accessibility to which is impacted | comma separated port(s) can be provided. If not provided, it will induce network chaos for all ports |  |
| DESTINATION\_PORTS | ports of the destination services or pods or the CIDR blocks(range of IPs), the accessibility to which is impacted | comma separated port(s) can be provided. If not provided, it will induce network chaos for all ports |  |
| PODS\_AFFECTED\_PERC | The Percentage of total pods to target | Defaults to 0 (corresponds to 1 replica), provide numeric value only |  |
| LIB | The chaos lib used to inject the chaos | Default value: litmus, supported values: pumba and litmus |  |
| TC\_IMAGE | Image used for traffic control in linux | default value is gaiadocker/iproute2 |  |
| LIB\_IMAGE | Image used to run the netem command | Defaults to litmuschaos/go-runner:latest |  |
| RAMP\_TIME | Period to wait before and after injection of chaos in sec |  |  |
| SEQUENCE | It defines sequence of chaos execution for multiple target pods | Default value: parallel. Supported: serial, parallel |  |