Transparent Caching Ecosystem

Introduction:

This documentation intended to be very brief and serve the purpose of giving high level architecture description only. Please refer to each component design document for extensive detail level description such as communication messages format, ladder diagram and state machine.

Introduction:

This documentation is work in progress living documentation.

Convention/Requirements:

* Microservice architecture.
* Every component should be decoupled from each other. There is no order for starting/stopping the services. Everything should work without specific order.
* Rotating log files will be stored in /var/log/madeline/<component-name>.log
* Each service should use systemd and backward compatible to sysVinit
* This product should work with other http caching server. Default http caching server will be nginx.
* This product should work with other http caching server. Default supported OS will be CentOS 7.x.

Summary:

Madeline is out-of/in-band inline (realtime) transparent caching ecosystem, which performs intelligent OTT online video (MPEG-DASH/apple HLS/etc) caching based on regular expression or ip address/port tuple logic rulesets. It must be fast in order to perform intelligent routing/load-balancing and filtering of packets per flow pulled from linux socket kernel buffer (SKB) bypassing linux network stack. The packets are decoded and dissected from layer 2 to 4 while the payload is untouched and pushed from one service to another in pipelined manner. Filtering will be done on packet-processing and http processing components and in the end interesting packets will be redirected to the caching web server such nginx/httpd/etc.

Components:

* Interface Packet Processing Service (IPPS)
* Packet Payload Processing (PPP)
* Packet Handling Service (PHS)
* Management Server (MS)
* Caching Server (CS)
* Simulation (SIM)

Architecture:

<<<Figure1 here >>>

As we can see from above figure, the main components are Packet

1. Component descriptions:
2. Interface Packet Processing Service (IPPS):

IPPS operation will involve partial kernel and user space. IPPS pulls packets from SKB using pfring based on interface name within kernel space. First filtering being done on the packet layer by specifying libpcap filter syntax at the kernel level. After the packets are pulled, the packets go through series of filters on the user layer. A load balancer is also added to load balance the packets per flow based on 4 tuples (src/dst MAC address, src/dst IP address). For example, any interesting packets will be passed from one pipeline to the next from IPPS to PPP then to PHS. Any uninteresting packets packets are ignored. Future work might allow user level to tell kernel to ignore the flow.

1. Packet Payload Processing (PPP):

PPP operation only involves user space and acts as plugin to the IPPS with “HTTP processing” as the default plugin.

PPP can be null operation if no payload parsing to be done on the packet. PHS by default set to process HTTP packet. PPP parses, dissects and applies regex on the packet payload (L4 payload) at user space level. It performs regex matching based on configuration given and determines if the packet needs to be passed to the PHS or not.

Example of PPP acting as packet injection plugin operation:

* above L4 decoding
* above L4 packet filtering and routing
* session correlation

1. Packet Handling Service (PHS):

PHS operation only involves user space and acts as plugin to the IPPS with “packet injection” as the default plugin.

PHS can be null operation if nothing to be done on the packet in the case “passive” monitoring mode. In case of “active” monitoring mode, it will inject and route packets as specified. The mode of operations is fictitious to illustrate that the PHS can serve as packet router with storage backend consuming the packets instead of transparent caching solution. PHS by default set to inject packet. In this case, if interested packets are received from the PPP, packet injection will inform the video server to route the traffic to caching server while at the same time stop the current traffic flow.

Example of PHS acting as packet injection plugin operation:

* RST packet generation
* Redirection packet generation
* RAW packet vector injection

1. Caching Server (CS):

CS will download content first time the server sees video traffic by asking the origin server for the content. After the first time, the caching server will just replay the traffic from the caching server.

1. Management server (MS)

MS manages configuration, registration and other management related operation. It serves as REST API central endpoint entry point performing the following operations:

* Configuration management.
* Endpoint registration and discovery.
* Health-wellness.
* Resource statistics

1. Simulation

TBD

1. Communication channels:

* Low-performance communication through AMQP (rabbitMQ)
  + Configuration
  + Registration/discovery
  + Health-wellness
  + Resource statistics
* High performance communication through ZeroMQ
  + One direction communication from pp-http to PHS
  + inter-thread communications within pp-http/PHS
* Management communication through REST API
  + User input REST commands to the management server

From figure1, IPPS, PHS and CS first registers with MS server, after the first initiation the MS server will periodically query all of them through AMQP communication channel.

Inter Process Communication (IPC) channels:

* IPPS communicates with MS server through AMQP rabbitMQ. The communication channels are bi-directional.
* PHS communicates with MS server through AMQP rabbitMQ. The communication channels are bi-directional.
* CS communicates with MS server through AMQP rabbitMQ. The communication channels are bi-directional.
* IPPS communicates with PHS through ZeroMQ. The communication is one direction.

1. Future work:
   1. In-band solution
   2. Web-caching server adapters (nginx, httpd, apache traffic server, etc)
   3. https (TLS) packets capturing with datastore/openstack on the backend
   4. network analytics
   5. ?
2. <<Reserved>>