Affinity Metadata Service Proposal

Name

Affinity Meta-Data Service

Repo Name

affinity

Description

The Affinity service provides an API to allow controller and higher-level applications to create and share an abstract, topology and implementation independent description of the infrastructure needs, preferences and behaviors of workloads that use the network to "talk" to one another. The term affinity has been chosen to recognize that many interactions between workloads are not random and unpredictable, but have known patterns that users, developers and operators are aware of. One way to think about this use of the term affinity is that it describes the nature of “conversations” across the infrastructure and their participants. The Affinity API allows intent to be specified in application and service terms independent of how and where the communicating workloads attach to the network, with no details of forwarding devices, vendors or technologies included. This can enable users to, for example, customize the behavior of virtual networks to better support their applications without knowing anything about bridges, routers, VLANs or tunnels. SDN Controller services and controller applications can use the affinity information to decide how to program data plane devices and can provide troubleshooting and monitoring outputs in the context of the conversations that users care about.

blocked URL

The Affinity concept is based on a layered architectural model in which intelligent controllers dynamically provision data center network infrastructure to satisfy workload affinities, as directed by external applications. (See figure 1.) The model features an abstract API that shields programmers from the nuances and complexities of the underlying infrastructure, so they can develop applications more quickly and easily. Designed for ease-of-use, flexibility and portability, it employs a small number of simple concepts, and provides for a wide range of implementation options.

blocked URL

The Affinity model distills the information exchanged between the application layer and the control layer to the smallest possible set of simple primitives: Affinity Identifiers, Affinity Groups, Affinity Links, and Affinity Attributes. Programmers can easily define affinities by using these four straightforward building blocks. Figure 2 summarizes and graphically depicts the fundamental Affinity concepts.

Figure 2. Affinity Constructs

blocked URL

Affinity Identifier This is how one specifies the traffic that is related to a particular "conversation". The identifier types can be extended for any conceivable method of specifying traffic. One type of identifier is directly visible in the network, like a MAC address that can be matched in an Ethernet packet header. Another type could be a reference to an object that will resolve to something in the packet. For example it could be a virtual NIC device name "vm235" that can be used to lookup a MAC address or tenant ID that can be matched during packet classification in the switched data path.

An Affinity Identifier represents a specific participant or group of participants in an Affinity Conversation. The identifier could delineate all of the traffic from a physical or virtual device, some of the traffic associated with such a device, or the aggregate traffic from a collection of devices. It could identify a bare metal server with a single Ethernet port, a storage appliance, a virtual machine, a port on a multiport NIC, or all of the members of a virtual network. More examples of Affinity Identifier types include MAC address, IP address, IP multicast group, VXLAN Network Identifier (VNI), and IP DSCP identifiers, etc.

Affinity Group An Affinity Group is a collection of one or more Affinity Identifiers, or other Affinity Groups. As the name implies, it allows objects to be grouped together so that a common set of Affinity Attributes can be applied to all of them. For example, one could create a “servers” group using Ethernet MAC addresses as Affinity Identifiers, a “storage” group using IP addresses as Affinity Identifiers, or a “virtual network X” group using a single VNI as the Affinity Identifier.

The Affinity Group construct is intended to be highly flexible. Groups can be composed of diverse resources (e.g. a tenant group and an application), a resource can belong to multiple groups, and a group can include any arbitrary collection of groups or individual objects. Affinity groups may be used to model a variety of common data center scenarios (e.g., a group of tenants sharing a data center, or participants within an application that have affinity).

Affinity Link An Affinity Link is a representation of a unidirectional relationship between a communicating pair of end-points that belong to one or more Affinity Groups. The Affinity Link allows Affinity Attributes to be assigned to communication flows between or within Affinity Groups. Affinity Links are unidirectional so that applications with asymmetric traffic flows, and other complex workloads can be accommodated. For example, a streaming media network might consist of a “servers” group and a “clients” group. The network traffic patterns from client to server are radically different from those between server and client. (The vast majority of the traffic flows from the servers to downstream clients) To make optimal use of network resources, two unidirectional Affinity Links are defined: one from the “servers” group to the “clients” group, and one from the “clients” group to the “servers” group. Distinct Affinity Attributes are assigned to each Affinity Link. Affinity Links can also be used to assign common Affinity Attributes to a group of end-points such as members of a VLAN or VXLAN. For example, the Isolation Affinity Attribute (indicates that traffic should be forwarded using separate physical links or paths not shared by other traffic) could be applied to members of a particular VLAN to provide network path isolation or separation within a shared multi-application or multi-tenant environment.

Affinity Attribute An Affinity Attribute describes the network properties that are assigned to an Affinity Link to meet workload performance, service quality, security, or other requirements. For the workloads communicating over an affinity link, these attributes are implemented in a manner not specified by the affinity abstraction.

Examples of Affinity Attributes include:

- Hop Count Sensitive - Assign to the shortest available path (not all traffic needs this)
- Isolate - Assign to an exclusive path not shared by other traffic
- Encrypt - Apply end-to-end encryption between the workloads
- Class of Service - Connect workloads using e.g. "silver" class of service
- Path Intercept - When sending traffic between group A and group B, insert a graph of paths between virtual functions

Examples of conversations that benefit from affinity-based forwarding:

- Preventing noisy neighbors in a shared data/storage LAN. Create a “storage nodes” Affinity Group that includes the addresses of all of the storage nodes. Create a link for this group and give it the isolate attribute. Next create a “storage clients” Affinity Group with the addresses of all the non-storage nodes. Create a pair of bidirectional links between the “storage nodes” and the “storage clients” Affinity Groups and assign the isolate attribute to both of these links. The SDN controller implementation could program the forwarding devices to put the client-server traffic on a different set of links from those assigned to the intra-node traffic. This would insure that activity on the intra-node links cannot possibly contend with traffic on physically separate multipath links between the storage clients and the storage cluster.
- Limiting latency between web front-end and back-end systems. Create a “Web Servers” Affinity Group with the addresses of the front-end servers and a separate “database servers” Affinity Group with the addresses of those systems. Create two links between the groups and assign both the “hopcount sensitive” attribute. The SDN controller could assign the server-database traffic to short paths and, if necessary, move traffic not identified as hopcount sensitive to longer paths.

Motivation

One of the most powerful capabilities offered by SDN is to use well-chosen abstractions to encapsulate complexity and allow different approaches to delivering the abstract functionality. This project would build the foundation for encapsulating the device/port/interface/media specific building blocks at the device layer, and present a more abstract interface option. We believe that it is in the interest of both network solution vendors and network operators to use a common approach to describing the network requirements of applications, using an extensible API that can encompass the wide diversity of forwarding technologies and use cases. One of the primary motivations is to allow applications to express network requirements in a common, less-networking-specific vernacular. The intent is not to re-create policy (VLANs and ACLs), though these abstractions might include policy-like elements. The ultimate goal is to create a workload abstraction that can be applied via different affinity attributes to different elements in the IT infrastructure. We believe that SDN logic residing on the controller should then translate topology and implementation independent requirements into low level device-centric rules.

Potential Controller Based Clients of Affinity Service:

- Topology Manager
- Forwarding Manager
- Security Manager
- User Manager
- HostTracker
- Routing Manager
- Overlay/Virt Manager
- Switch Manager
- New control plane modules

Potential Sources of Affinity data:

- Cloud Orchestration systems
- Management apps
- Monitoring/analytics systems
- Service Assurance systems
- Virtualization managers
- Self Service Portals
- Policy Managers

Benefits of Affinity in ODP

- Allows Northbound entities (CMS, Orchestration, Enterprise Applications) to describe workload needs in terms of service level, rather than per-device configuration.
    - Simple REST API allows import of rich affinity information
- Affinity attributes can be exposed in user interfaces to allow infrastructure customization by users who are not network administrators or engineers.
- Changes the network consumption model to a self-service, on-demand approach tailored to cloud traffic patterns

- Control Plane Logic gains access to the explicit needs of the workload communications and thus can automate optimization of resource usage
- Enables infrastructure to adapt to applications, rather than the reverse
- Hide some of the complexity of device level configuration, provisioning, and management from the layers above.

The scope of the Affinity Service includes:

- Definition of a common, extensible API
- RESTful API implementation with CRUD operations for all identifiers, attributes, groups and links
- Language bindings for REST API comparable to other controller modules
- Code to populate the data model objects, preserve relationships between them, and share them among modules.
- Persistent storage for object repository
- OSGI package implementation
- Sample client code and affinity based optimization examples

Current Status

Presentation and Demo

1. File:Opendaylight affinity API.pdf **Slides from the Nov 21 presentation** to the OpenDaylight TSC meeting.

The affinity service comes with a demo of the waypoint redirection service, maintained in the file scripts/demo.py. To run the demo:

1. Check config.ini for two things: First, that all of the affinity jars are getting loaded. Second, that of.flowStatsPollInterval is set to 1, not 10.
2. Start the controller. Once it's up, you can run `ss | grep affinity` to make sure all bundles are loaded and ACTIVE.
3. Start demo.py. Importantly, this needs to happen before you run mininet, so it can add the subnet.
4. Step 3: Start mininet.
    a. SSH into the VM
    b. Launch the topology. `sudo mn --controller=remote,ip=<controller ip> --topo tree,2`
5. Mininet commands
    a. `pingall` (The hosttracker needs to know about h2 in particular because we're going to use it as a waypoint). Once this command has completed, you can press "enter" in demo.py to set up the redirection and per-protocol flows (the per-protocol flows won't be installed correctly unless there are already flows in the network)
    b. `h3 ping h1`. demo.py is set up to track large flows into the subnet 10.0.0.0/31, which in this topology, only contains the host 10.0.0.1. So we want to send traffic into that host.

You'll see an anomaly detected in demo.py's output. It will automatically create two affinity groups: one out of the prefix 10.0.0.0/31 (the prefix we're targeting), and one containing 10.0.0.3 (the host sending the most data). It will create a link between the two, and enable waypoint redirection to 10.0.0.2. You will also see the pings stop immediately. If you run the pings in the background, and run tcpdump on h2, you'll see h2 receiving the ICMP pings.

You can also disable the waypoint in the demo (it will prompt you to press enter to do so), and then the pings between h3 and h1 will resume, and traffic will no longer be redirected to h2.

Scope

The scope of the Affinity Service includes:

- Definition of a common, extensible API
- RESTful API implementation with CRUD operations for all identifiers, attributes, groups and links
- Language bindings for REST API comparable to other controller modules
- Code to populate the data model objects, preserve relationships between them, and share them among modules.
- Persistent storage for object repository
- OSGI package implementation
- Sample client code and affinity based optimization examples

Current Status

Presentation and Demo

1. File:Opendaylight affinity API.pdf **Slides from the Nov 21 presentation** to the OpenDaylight TSC meeting.

The affinity service comes with a demo of the waypoint redirection service, maintained in the file scripts/demo.py. To run the demo:

1. Check config.ini for two things: First, that all of the affinity jars are getting loaded. Second, that of.flowStatsPollInterval is set to 1, not 10.
2. Start the controller. Once it's up, you can run `ss | grep affinity` to make sure all bundles are loaded and ACTIVE.
3. Start demo.py. Importantly, this needs to happen before you run mininet, so it can add the subnet.
4. Step 3: Start mininet.
    a. SSH into the VM
    b. Launch the topology. `sudo mn --controller=remote,ip=<controller ip> --topo tree,2`
5. Mininet commands
    a. `pingall` (The hosttracker needs to know about h2 in particular because we're going to use it as a waypoint). Once this command has completed, you can press "enter" in demo.py to set up the redirection and per-protocol flows (the per-protocol flows won't be installed correctly unless there are already flows in the network)
    b. `h3 ping h1`. demo.py is set up to track large flows into the subnet 10.0.0.0/31, which in this topology, only contains the host 10.0.0.1. So we want to send traffic into that host.

You'll see an anomaly detected in demo.py's output. It will automatically create two affinity groups: one out of the prefix 10.0.0.0/31 (the prefix we're targeting), and one containing 10.0.0.3 (the host sending the most data). It will create a link between the two, and enable waypoint redirection to 10.0.0.2. You will also see the pings stop immediately. If you run the pings in the background, and run tcpdump on h2, you'll see h2 receiving the ICMP pings.

You can also disable the waypoint in the demo (it will prompt you to press enter to do so), and then the pings between h3 and h1 will resume, and traffic will no longer be redirected to h2.

Resources Committed (developers committed to working)

- Derick Winkworth (Plexxi)
- Kevin Tronkowski (Plexxi)

Initial Committers

- Derick Winkworth (Plexxi)
- Kevin Tronkowski(Plexxi)

Vendor Neutral

New Project. No Code

Meets Board Policy (including IPR)

New Project. No Inbound Code Review needed.