Transit forwarding within an AD

How is the packet forwarded inside the domain when it is transiting through the domain? Traditionally, this was done in two ways:

1. Installing forwarding entries to other ASes in the interior routers.
2. Encapsulating the packet at the boundary router to forward it to the next hop boundary router (i.e., egress router of the AS). MPLS is generally used for forwarding within the AS.

AD forwarding table in interior routers

Packets are forwarded as-is inside the domain. The interior routers possess forwarding entries for each AD, and packets are forwarded by exact matching of each node in the DAG, including AD. This would be the cleanest solution to implement.

Advantages

- No need for extra logic in processing packets at the boundary routers.
- Gives more control to the AD in handling transiting flows. For example, it can service some flows via a longer and less loaded path. (Note: This can also be achieved in the encapsulation based forwarding, by introducing a label based switching and distributing the label forwarding entries in the interior routers.)

Disadvantages

- Every router needs to know how to forward based on next-hop AD, so table size will be larger.
- The controller needs to compute AD forwarding tables for the interior routers, and distribute it to them.

Encapsulation: Solution 1

Packets are encapsulated at the boundary router with destination next-hop, which is the egress router. The interior routers are unaware of the AD forwarding semantics. They forward based on the destination in the encapsulated header, viz. the host ID of the egress router.

Advantages

- Smaller forwarding tables in interior routers. They don't need have the AD-based forwarding table.
- For current Internet, this is traditionally implemented using label based switching (MPLS) which involves exact matching instead of longest prefix matching. Exact matching for labels is faster and cheaper than longest prefix matching for IP addresses.
    - For XIA, however, AD routing only requires exact matching. This is not really an advantage in the XIA context.

Disadvantages

- This needs more complicated logic at boundary routers.
- Encapsulation capable routers at the border. (This may not be a big disadvantage.)

Example - Simple

1. Boundary router receives a packet with destination DAG: ●→ AD(x) → HID(y)
2. It resolves the egress router host ID (say HID(egress)) corresponding to AD(x) from the routing information.
3. It encapsulates the packet with destination DAG: ●→ HID(egress) → Decap, and sends the packet to be routed inside the domain.
4. The egress boundary router receives the packet. While processing the destination DAG, it figures out based on Decap that it has to decapsulate the packet, and further process the packet.
5. Based on the AD forwarding table, the egress router forwards the decapsulated packet to the correct next-hop AD.

●→ HID(egress) → Decap
●→ AD(x) → HID(y)

(underline indicates that the *next-node* pointer is on this node)

Example - Complex

Since XIA permits forwarding packets with a more complex intent than the one discussed above, the processing logic in the ingress boundary router could be more complex. For example, a packet with destination DAG: ●→ AD(c) → SID(b) → AD(x) → HID(y) will be a non-trivial case. The ingress router in AD(c) has to process the DAG till an AD node (i.e., AD(x) in the example) is encountered, and the encapsulation should preserve the intent of the original destination DAG, which is to pass through SID(b) before exiting AD(c).
In this case, the encapsulated destination DAG will be ●→ SID(b) → HID(egress) → Decap. The *next-node* pointer in the original DAG needs to be updated to AD(x), since when the packet reaches the egress router, it needs to start processing from the AD node in the DAG.

●→ SID(b) → HID(egress) → Decap
●→ AD(c)→SID(b) → AD(x) → HID(y)

(underline indicates that the *next-node* pointer is on this node)

Implementation

1. The ingress router extracts a subgraph from an incoming packet's DAG
    a. The DAG is parsed up to the node representing the next AD in the DAG, or to the final destination if no AD is present.
    b. This is the subgraph of the DAG that the AD will make use of in routing within its domain.
    c. e.g. For ●→SID→ AD→HID, first two nodes are processed to obtain ●→SID→AD to encapsulate the packet.
2. The *next-node* pointer is updated to point to the next AD in the DAG.
3. A copy of the subgraph is made, with the AD node being replaced by the HID of the egress router
    a. e.g. ●→ SID → HID(egress)
4. A node is appended at the end to indicate that the packet should be decapsulated.
    a. e.g. ●→SID→HID(egress)→DECAP

5. At the egress router, the packet is decapsulated and the header is reverted to the original form.

Encapsulation: Solution 2

Same as solution 1, except complex DAG translation for encapsulation is avoided, by hosting services in the edge network. The edge network will be able to lookup egress HID for an AD.

●→ HID(egress) → Decap
●→ AD(c)→SID(b) → AD(x) → HID(y)

Disadvantage

There is a constraint that the services can be hosted in the edge network of the AD.

Bootstrapping an AD

What should be statically configured to allow routers and controllers within an AD to identify each other?
This static configuration will be used to bootstrap the network connections, allowing controllers to communicate with routers to create a coherent routing policy. We approach the problem from two angles, corresponding to whether the topology is initially known.

Static configuration of topology

The most straightforward method of bootstrapping an AD is to configure the topology statically, as is done in most current SDNs. One method is to have topology configured at all network elements. Another is to only configure topology at the controllers, but also let routers know which controllers to trust by giving each the public key of the controllers.

Advantages

- Quick bootstrapping phase.
- Controller can calculate and disseminate routing tables immediately.

Disadvantages

- Inflexible -- not resilient to network changes and not as maintainable.
- More work in configuration stage -- possibility of human error.

Dynamic discovery of topology

With dynamic discovery, the network elements are able to act independently to arrive at a consistent and desired set of actions for the entire AD.

Requirements

To enable dynamic discovery, there are two main requirements that must be met for correctness, performance, and security

- Each router has to know which controllers manage it.
    - Prevents other ADs' controllers from taking control of it.
- Each controller has to know which routers it is managing.
    - Avoids route calculation being distorted by routers posing to be from the same AD.
    - Prevents adversaries from being able to extract sensitive information regarding internal network topology.
        - e.g. by appearing to join the AD at different points in its internal network

Method 1: All controllers and routers possess the public/private keys of the AD

With this method, all routers can self-identify as boundary routers if they detect that a neighboring router belongs to another AD. Routers signal their AD by sending HELLO messages consisting of a digest signed with the AD private key, which can then be verified with the AD public key. Once external-facing interfaces are established, routers can simply flood LSA on internal-facing interfaces to bootstrap reachability within the AD.

Advantages

- Can adapt to network changes and easier to maintain
- Simple to implement
- Semantically consistent -- notion of identity is equivalent to possession of private key

Disadvantages

- Potentially risky, as private key may be exposed if any one element is compromised
    - One solution is to keep the private key in the hardware, e.g. use cryptographic coprocessor

Method 2: All routers possess controller public key and controller possesses all router public keys

With this method, both controllers and routers are explicitly configured with which network elements they can trust. The controller builds out the graph of AD connectivity one hop at a time, by sending out HELLO messages with digests signed by its private key. The first hop switches are able to verify the message source, and ACKs with their public keys and a message signed by their private keys, which the controller then checks. The controller then updates the router's routing table. In subsequent iterations, the controller instructs the furthest routers to repeat the process, sending the signed HELLO messages out their ports and relaying the replies back to the controller. The discovery algorithm eventually terminates at boundary routers, since the next hop routers after that will not be able to produce the correct ACK message.

An optimization to reduce the amount of configuration required is to only give the controller the public keys of boundary routers. In the absence of configuration errors, the controller can safely assume that all routers reached before the boundary belong to its AD.

Advantages

- Can adapt to network changes and easier to maintain
- No sharing of private keys

Disadvantages

- Slower discovery of network topology, since it is done hop by hop starting from controller