Melwyn Pak
CS1680 Capstone

Equal-Cost Multi-Path (ECMP) Routing in Software-Defined Networking

What is ECMP

ECMP is a simple load-balancing routing scheme for networks. Instead of having a single "best" path (measured by some metric such as number of hops) to a particular destination, ECMP allows for multiple "best" paths where possible. This enables some form of load-balancing as we can spread traffic across all paths if necessary.

Introduction to SDN

In regular networks. switches are "intelligent" in the sense that each switch is capable of running some routing scheme such as a distance-vector algorithm to create a local routing table. However, these switches are usually costly. Furthermore, since the routing table of each switch is constructed in a distributed manner, it is hard to debug or understand how a best path was calculated. Software-defined networking (SDN) was introduced to solve these problems. By decoupling the control plane (i.e the portion of the network functionality that decides where to route packets) from the data plane (the portion of the network functionality that forwards packets), SDN makes it easier for network administrators to control and debug routing schemes. Besides that, since most of the control plane functionality is now abstracted into centralized controllers (think of them as servers), switches in the network no longer need to be "intelligent" and therefore cheaper switches can be utilized.

When using SDN, the SDN controllers possess an omniscient view of the entire network. As such, we can represent the network as a graph where each switch is a node in the graph and the edges are the physical links between switches. Consequently, we can apply shortest-path algorithms such as Dijkstra or Bellman-Ford to calculate the shortest path between any pair of nodes. Once the global routing table has been constructed, the controller then installs routing rules in each of the switches in order to enforce the routing table.. A typical rule is of the form <*MatchingCriteria*, *Action*>. *MatchingCritera* specifies how the switch should identify the incoming packet, for example the switch should attempt to match rules based on the destination MAC address of the packet. *Action* on the other hand specifies how the switch should handle the packet. This is usually an instruction to the switch to forward the packet out on a particular port. If a particular packet does not match any rules, the default behavior is for the switch to send the packet directly to the controller.

Implementing ECMP in SDN

By implementing ECMP, if there are multiple best paths to get to a particular MAC address (i.e. more than 1 path has the same cost metric), the controller installs rules in switches such that all the paths are utilized as opposed to just picking 1 path for that MAC address. This is accomplished by specifying rules that match on more fields in conjunction with the destination MAC address. For this implementation, I used Floodlight[1], a popular Java-based SDN framework. Furthermore, the switches in this implementation run Open vSwitch[2] which implements OpenFlow[3], a protocol for communicating with switches and SDN controllers.

Implementation details

1. Each switch in the network is represented as GraphNode object.
2. Each switch keeps a list of minimum cost paths to a particular MAC address as opposed to just a single path. This requires a slight modification to Dijkstra's algorithm. When we discover a new path of a lower cost than what the switch currently knows, the entire list of minimum cost paths is discarded. However, when we discover another path that is of the same cost as existing paths, we add it to the list (as opposed to ignoring it in the canonical Dijkstra's algorithm). The list of paths and collection of rules in each switch are hard-state i.e they do not timeout and are explicitly cleared.
3. Whenever a change in the topology is detected, all rules in the switches are cleared and the shortest paths are recomputed and cached in each GraphNode object. However, the rules are not installed yet.
4. Rules are only installed for flows that are seen by the controller. The exact steps are as follows:
    - A packet arrives at a switch. Since the switch does not have rules that match with it, the switch forwards the packet to the controller.
    - The controller then checks to see if the switch has more than 1 best path to the packet's destination MAC address. If it has, it then picks a path based on the least used port. The rule installed would then match on the source and destination MAC address. This means that a packet to the same destination but from a different source could be routed to another equally good path if needed. If there is only 1 path from the switch, then the controller installs a rule that matches only on the destination MAC address.
    - Once the rules are installed, the controller forwards the packet on behalf of the switch to the next hop.
5. Each GraphNode keeps track of the number of times each of their ports are used in a rule. This allows the controller to decide which port is least used. This is required to

[1] http://projectfloodlight.org/
[2] http://openvswitch.org/
[3] https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

guarantee fairness in the case of flows from a single source MAC address to different destination MAC addresses. Without keeping track of this, it might be the case that the controller would install rules such that packets destined for host H1 and H3 will always be forwarded out of port 1 even though there is another equally good path to either of those hosts out of port 2. By ensuring that the controller only picks the least used port, packets destined for H1 would be routed out of port 1 but packets destined for H3 would be routed out of port 2 instead.

6. As mentioned in #2, rules never timeout. Instead, whenever a the topology of the network has to be recomputed, the controller clears all the rules in every switch.

Link to code: https://bitbucket.org/mp42/cs168_sdn. Due to academic regulations, the repository is private. For access to the code, contact me at melwynpak at gmail dot com.