How to calculate DPC in a distributed way on top of BGP

A brief overview of how to extend BGP to calculate the DPC in a distributed way

Assumptions

I assume that the reader already knows what the (Destination Partial Centrality)DPC is and how BGP V4 works.
An explanation of DPC is given at the end of the file.

Modifications on top of BGP to calculate DPC

It has been possible to implement the distributed calculation of DPC on top of BGP thanks to some aspects already inside the protocol.
For the implementation, we will follow works already done in this field like [1][2].
First of all, we will use the UPDATE messages to calculate the centrality metrics, this because those messages are the one that will share the knowledge in the network.
We will extend the section “Path attributes” of the update message with some new optional-transitive messages, like defined in the RFC4271 section 5 [3], this will ensure us the fact that legacy and new BGP speaker can coexist at the same time.
The two new attributes are described by the table below:

Attribute name	Composition	
NH	NH_Number (2 octets)	NH_AS (variable)
ASLoadList	AS_Load_Number (2 octets)	AS_ID + AS_Load + AS_State (variable)

- NH: this attribute represents all the possible NHs that it can choose to reach the destination with the shortest path. It represents what in the definition of the DPC is “potentially multiple shortest paths”, the first part contains the number of NHs in 16 bits, and the second one is the list of NHs, each of them is a single parameter of 32 bits, so the total size of this attribute is of (16 + 32*NH_Number) bits;
- The second one is the ASLoadList know by the sender of the UPDATE, the sender is self-included. Like before the first attribute represents how many objects are in the second one, and the second one is a list of load metrics. Each load metric is a composition of the AS identifier, the actual load corresponding to the identifier and a parameter used to know how much is updated this information. This is the parameter that will distribute the centrality knowledge

Noticeable points

This attributes merged with the algorithms (applied on top of BGP with minus changes) already presented in [1][2] will make possible to calculate the DPC, but we have to be careful about some aspects.
By the definition of DPC, we know that only the nodes that share a destination will actively participate to the protocol, only those node will increase the input commodity by one unity, other nodes (transit nodes) will not increase the commodity.
This will ensure to respect the two main sets of the DPC.

Additional data structures

Each BGP node should maintain a set of additional information to be able to calculate its centrality. In the routing table we are not going to have just the destination and the next hop, but also the input load for this destination, so a new “load” attribute is appended to the RT.
In this way, each node will be able to keep count of how much commodity will pass through it to reach the destination.
And other than that, a node has to keep a dictionary with the most updated centrality of all the nodes that it knows, and its own centrality can be in this dictionary.

Node behaviour

At the beginning, a node should initialize its own data structures empty and in the dictionary, there is just its own line with an associated centrality of 0.

At the reception of an UPDATE, the node has to process the information to update its own input load, update its knowledge of other nodes centrality.
Its own input load is given by the sum of the load shared by nodes that have chosen it like NH for the destination, and the dictionary is going to be updated only if a new or a more updated line is found in the ASLoadList attribute.
The evaluation of those parameters inside a new UPDATE message just received should be evaluated only if the UPDATE message is useful, if policy filters throw away the message is useless to evaluate the new information.

A node is responsible for the update of its load, when it changes it has to increase by one the AS_State of its load in the dictionary, so when other nodes will receive this information they will notice that a more recent value is available and they will update their own dictionary.

When a node has to share an update, first of all, it will calculate its own loadout, if it is a transit node the loadout it is not going to be increased, otherwise, it will add one commodity unit to the output.
To the updated will be included also the NH and the ASLoadDict.
Is possible to implement different techniques for the ASLoadDict sharing, we suggest sharing just the newest/updated pieces of information to not overload the argument.

Conclusion

Is possible to find a Bird modification with the distributed calculation of the DPC at the following link:
https://ans.disi.unitn.it/redmine/projects/internet-on-fire/repository/iof-bird-daemon?utf8=%E2%9C%93&rev=milani%2Fbird-2.0.1

DPC explanation

We take into account a graph $G = (V, E)$ that describes relations between Autonomous Systems (AS)s. Some ASs share routes, a route is a pair $r = (d, \xi)$, where d is the destination and ξ the set of attributes associated with the destination.

One of the attributes is the "originator node" $O_n \in \xi$, this attribute identifies uniquely the node of V that originate the route d. Each node that exports at least one route is said to be a destination node; the set of all destination nodes is described with $C = \{j \in V : \exists r, r[O_n] = j\}$. The opposite set is $T = V \setminus C$ and represents all the nodes that do not originate any route, but they only act like transit nodes.

Taking a node i we describe with N_i, the number of routes that the node i export.

The only nodes that contribute to the load calculation process are the destination nodes, but each node have a load that is calculated with the input contributes by other nodes.

Is now possible to define the Destination Partial Centrality (DPC) like following:

definition A1.1: DPC adapts load centrality to represent the propagation of routes in an IP network. In DPC load represents the amount of networks that a BGP node exports. Not all nodes that run BGP generate load, but all nodes that forward traffic have a non-zero DPC centrality. We call $C \subseteq V$ the set of nodes that can be a source and/or destination of traffic (they export at least one network) and N_s, N_d the number of networks that are exported by node s and d, respectively, then $\theta_{s,d} = \frac{N_s + N_d}{2}$, DPC $\Delta(v)$ of any vertex $v \in V$ is defined as:

$$\Delta(v) = \sum_{s,d \in C} \theta_{s,d}(v)$$

we assume that $v \neq s \;\&\; v \neq d \;\&\; s \neq d$.

There is a special case to consider: $C = V$ in this case, all nodes are in the destination set, so the opposite set $T = \varnothing$ is empty. In this case, we fall back into the Load Centrality (LC)[3] and $\Delta(v) = LC(v)$.

DPC could be also be normalized with the following equation:

$$\overline{\Delta}(v) = \frac{1}{|C|(|C|-1)} \sum_{s,d \in C} \theta_{s,d}(v)$$

References

[1] L. Maccari and R. L. Cigno, “Pop-routing: Centrality-based tuning of control messages for faster route convergence,” pp. 1–9, 2016.
[2] L. Maccari, L. Ghiro, A. Guerrieri, A. Montresor, and R. L. Cigno, “On the Distributed Computation of Load Centrality and Its Application to DV Routing,” in IEEE International Conference on Computer Communications (INFOCOM), Honolulu, USA, 2018.
[3]U. Brandes, “On Variants of Shortest-Path Betweenness Centrality and their Generic Computation,” Social Networks, vol. 30, no. 2, pp. 136–145, May 2008.