Features of intent based specification:

1. Conflict detection: While specifying policies through this tool, we don’t need to think about conflicts as this framework easily detects conflicts. Also, in case of conflicts between different level policies, it would be even more difficult to detect such conflicts without this tool.
2. Duplicate policies: There may be so many duplicate policies which may become a bottleneck in network performance. Also, it is easier to manage policies if we have a minimalist representation of the policies. As in case of conflicts, it would be hard to find duplicate policies also.
3. Ease of specification: It is very to specify policies in a short and concise manner. Also, it is very easy to extend the structures and policy types which give the user a lot of flexibility to suit their infrastructures.
4. Composition: It is easier to co-ordinate and compose different policies amongst different administrators and to detect conflicts across different policy specified.
5. Dynamism: Network policies may be very dynamic and it might be needed to change them very often. It may be taxing to go through earlier specified policies for conflicts and then specify new policies. With this framework, it would be easier to update and delete policies frequently.

# Normalization:

Bringing all source and target devices of the same abstraction type to the same level is very important to capture policies in the most logical manner. As we are capturing the policies in a single graph, it will be totally impractical to check for the conflicts and repetition. To understand it better, let us consider an example. If we have a policy for Nest cams and another policies for all nest devices then if we are capturing these two types of devices at different levels then we have to go through all the devices and for each combination, we would need to check possibly for all the levels in that abstraction tree to find out the possibility of conflicts. This process would become very costly once the number of sources and targets increases even to a hundred devices (O(nk) k=levels of abs tree and and n = number of sources and target devices ) . To handle this better we first find out the lowest level for a certain abstraction types (iot-devices etc.) and then bring all devices to this level before storing the policy for a source. This helps in maintaining the policy applicable to a device together and saves us unnecessary checking of over all the devices.

Suppose there is a scenario in which we have a gateway available to enforce policies only at a certain level of abstraction then we can define the **elevel (**enforceable level) of abstraction and bring policies down to only this level. If there are certain policies defined for devices below the elevel then we define a plain list of such policies and enforce it using different measures.

# Normalization Algorithm:

Function set\_min\_level (policylist,abstract,lowestlevels):

1. For policy in policylist:
   1. For policytype in policy:
      1. Parse policy to find out the source and target for the policy and the abstraction type
      2. Set Source\_level := len(source\_parent.split(“->”)) and target\_level := len (target\_parent.split(“->”))
      3. If Source\_level > lowest\_level(source\_abstraction\_type) or lowest\_level(source\_abstraction\_type) is not set:

Set lowest\_level(source\_abstraction\_type) := Source\_level

* + 1. If Target\_level > lowest\_level(target\_abstraction\_type) or lowest\_level(target\_abstraction\_type) is not set:

Set lowest\_level(target\_abstraction\_type) := Target\_level

1. For policy in policylist:
   1. For policytype in policy:
      1. If Source\_level < lowest\_level(target\_abstraction\_type):

Set Source\_nodes := all\_child\_nodes at lowest\_level(target\_abstraction\_type)

Else:

Set Source\_nodes := source

* + 1. Similar process for target to set Target\_nodes
    2. For snode in Source\_nodes:

For tnode in Target\_nodes:

If no existing policy for this combination:

Add Edge with snode and tnode and the source and target

Else:

Report conflict or duplicate between these and keep the older policy intact

# Composition Algorithm:

Function ComposeGraphs(Set\_of\_Graphs , Network\_Translations, abstractions):

1. For graph in Set\_of\_Graphs:

For node in graph:

Set node\_Network\_eq := Translations[nodes]

For node in graph:

If Set\_of\_network\_nodes(graph) contains node:

Check all edges with this node for conflict

Add\_edge\_to\_composed Graph

1. Repeat the same process for Composed\_graph

Function TranslateGraph(Composed\_Graph, ipmapping):

1. For node in Composed\_Graph:

For dest in Composed\_Graph[node]:

Srcip = ipmapping[Translations[nodes]]

Destip = ipmapping[Translations[Composed\_Graph[node][dest]]]

Add\_edge to final\_graph