Third Party Groups

Now that Groups is implemented, we are considering how to bridge a third party concept of Groups into Matrix.

The requirements are to bridge a third party **user** such that:

- The user is a member of a Bridge Group;
- The user's publicised groups include this group.

And to bridge third party **room** such that:

- The room belongs to the Bridge Group;
- The room contains a "m.room.related_groups" event, containing the Bridge Group ID so that Flair can be shown by clients viewing that room.

Solutions

<table>
<tr><th>Pros</th><th>Cons</th></tr>
<tr><td colspan="2"><h3>A. Bridge specifies a group to matrix-appservice-bridge</h3></td></tr>
<tr><td colspan="2">Configure the matrix-appservice-bridge Bridge object with a group ID that is used by the library to add users and rooms to a group at the point of creation. See Travis's https://docs.google.com/document/d/1MgoMNQXrPzUg3PTTrYanLCP-YpfZM25QxH0JnDmK4Cw/edit</td></tr>
<tr><td>- Would allow bridges using matrix-appservice-bridge to add users to groups and set related_groups of rooms.</td><td>- Existing users/rooms would still have to be added by the bridge itself (e.g. when it joins a new ghost to a room). This is because the bridge library would only add to groups when the third party network API is hit, i.e. when room aliases and user IDs are resolved (within the bridge's namespace).
- Doesn't consider other types of ASs.
- Flair would only be enabled in portals. The bridge would have to insert related_groups for other types of rooms.</td></tr>
</table>

B. Group Server delegates namespaced groups to AS

Delegate queries to an AS from the Group Server. This assumes that each call to the API can be resolved to a particular AS, potentially using some sort of namespacing on Group IDs (i.e. `+.irc.freenode:matrix.org`, where `freenode` could be an third party instance ID).

- No need to sync bridge data with HS, the one source of truth is the AS. - Considers all ASs. - AS can consider many instances, if it wants to.	- AS has to worry about being performant in the face of group queries. - AS has to worry about pagination.

C. Group Server (Magically) Does AS Groups

Preferred immediate solution for the problem of "put users from a given AS into a given group". It's orthogonal to option B, which we can move to in future. Meanwhile Option A is going to become a world of syncing pain so let's not do that.

Originally: "The Group Server is given some configuration to map a given AS to a group, potentially namespaced as with B. When a user is registered as an AS user on the HS, ping the GS to add it to the namespaced group. Similarly for portals."

In practice, we can just do this by adding a "group: '+wherever:matrix.org'" to the user namespace section in the registration for a given AS. The group server code in synapse will need to 'magically' pick up the existence of these groups from the AS registration config, and then specialcase the API responses such that users the given prefix are considered unilaterally to be members of that group. This is very much an HS-specific implementation feature rather than anything to do with the Matrix spec.

Separately, the ASes may well want to go and try to add the group to the 'related groups' state event for the rooms they are participating in.

As an additional bonus, we could also put a 'group' entry in the room namespace section in the AS registration, and have the GS magically maintain the list of rooms for a given prefix as being associated with that group. However, this then (comically) duplicates the existing 3PL RoomDirectory function, so we can ignore it for now (and separately, later, consider whether should replace the current 3PL API with this.)

- Considers all ASs. - A group per instance could be done by specifying one per namespace in the	- ~~Different instances would be tricky.~~ - It would be a bit strange to have the GroupServer set the related_groups

AS registration file.	state in a room. So the AS would have to do this still. - Other kinds of rooms (not portals) are not considered.

Conclusion

Having had further discussion with Erik, we've decided that option C is preferable but we needn't implement the API to list 1000s of group users as it would involve implementing pagination. The worry here is that we'll end up implementing every feature for groups twice. By limiting the scope to the API that determines a user's publicised groups, the implementation is much simpler as it will involve a trivial RegExp against the user's ID (no database/pagination involved).