On Entity Associations In A Cloud Network

January 18, 2010

In this note I am attempting to set down some ideas I have about persistent network relationships between entities that exist with a cloud-net.

Before launching into the main body of this note I thought I'd give conclusions first:

1. That cloud computing entities need names that can fit into an unobjectionable domain name "label" that is limited to the normal "hostname" constraints (i.e. alphanumeric plus hypen, 1-63 octets, no leading hyphen, etc)
2. That cloud computing entity names should be a textual representation of a 128 bit globally unique identifier, i.e. a GUID.
3. That cloud computing entity names need a sub-structure or two-part form so that the name of the entity in its entirety can be expressed as can the name of a particular localized point of contact of that entity.
4. That we need a layer above TCP that creates persistent associations between users of cloud entities and those entities. These associations ought to be able to endure changes of underlying IP addresses (and thus failure and re-establishment of TCP connections).
5. The association layer needs some sort of handshake so that the peers using the association can establish simple names that the peers can use as checkpoints to resynchronize themselves as the association reforms after the failure and re-establishment of underlying TCP connectivity.

Cloud Entities

When I say "entities" I mean something analogous to an application, without necessarily limiting that notion to something that exists purely in a client or server role. When I say *entity* I mean something that is above the TCP and UDP layers and more mobile and expansive than a traditional internet application.

Some of today's internet applications are striving towards what I mean as a cloud-net entity. They are doing so using an ad hoc collection of mechanisms ranging from stateful DNS servers, anycast routing, load-balancers. And in most instances these approaching-cloud-net applications use URIs as persistent or transient names.

Here are some of the characteristics that I think might adhere to a cloud-net entity:

1. The entity may have existence that spans several providers of lower level abstractions. By this I mean that there may be parts of the entity running in several data centers operated by different vendors. This suggests that the entity uses, at least internally, more than a single IP address and that those addresses are not necessarily confined to any one subnet or address block.
2. The entity may grow, shrink, or move so that it changes its use of lower level abstractions such as data centers or IP addresses.
3. Entities may suffer from partitioning of the lower layer network. (A side effect of this is that there may be times when two or more things on the net each legitimately claim to be "the" entity; this may complicate the issue of identification and authentication of entities.)
4. An entity may take time to achieve wide-scale consistency. This may make it necessary for a

user of the entity to be able to maintain connectivity to a the same part of an entity so that that user can obtain the benefits of local consistency within the entity.

Names and Addresses

I personally find attribute-based lookup systems, such as IF-Map to be intriguing as a tool for solving many of the issues that will arise in cloud based networking. However, it seems to me that at the end of the lookup process, whether it be IF-Map or some other system, we need to find a name or an address of some kind.

The domain name system has served us well. It seems to me that whatever kinds of names we might invent for cloud entities we ought to take care that the names can be used as labels in domain names.

Please do not take my hope that a cloud entity name can be a domain name label to mean that we should use DNS names as cloud entity names. I am merely suggesting that DNS is a well deployed and reliable tool that people will want to use in unforeseeable ways; it seems that cloud entity names that can be DNS labels will eliminate a possible obstacle to innovation.

In my mind I imagine DNS names that lead to a new kind of Resource Record that contains a cloud entity name.

That name could be used much as we use the contents of NAPTR resource records in ENUM, i.e. as data that is used to generate further query names that eventually lead to address (IPv4 or IPv6).

Given the trademark wars and language script issues that have battered DNS for the last decade, I would further suggest that cloud names be free of human-semantics. If someone wishes to create semantically meaningful names then I would urge that those semantically meaningful names be pushed into other systems outside the scope of our endeavors and that those systems should result in the kinds of semantically meaningless names described here.

I would suggest the “globally unique identifier” or GUID as the basis for cloud names. (See http://en.wikipedia.org/wiki/Globally_Unique_Identifier)

GUIDs are well documented, there are broadly accepted interchange formats, and they are easily handled in software. Textual representations can easily fit into the 63 octet limit on domain name labels and can abide by the “hostname” format limitations.

Two Tier Names

As I mentioned previously, cloud entities may temporarily be split into multiple disconnected parts as the result of partitioning of the underlying net services.

And either as the result of such a partitioning, as a side effect of normal network delays, or simply as a result of design, a cloud entity may not always be fully internally consistent.

A user who is performing data updates with the cloud entity may find the experience more satisfying if the results of those updates can be perceived by that user.

This suggests that users may not be happy if related connections to a cloud entity are scattered hither and yon across the various contact points that might be available.

Rather, a user might want to maintain a kind of locale relationship with the cloud entity. In other words, a user who is carrying on a dialog with a cloud entity might need to interact with the cloud entity through contact points that that are by some metric “close” to one another.

These issues suggest to me that the names (or addresses) of cloud entities need to be formed in a way so that peers that care have the ability to express the name (or address) in a way that will assure (not guarantee) that they will obtain an connection association that closely resembles some prior (and similarly named or addressed) prior connection association.

Of course, if the peer does not care about preserving some prior association then the name (or address) ought to have a form that elides the proximity part or wildcards it.

Thus a name (or address) of a cloud entity might be considered a tuple formed by a base name and a name that represents a sub-realm of the cloud entity in which a user can operate and perceive consistent results.

I would suggest that this sub-realm name be formed in the same way as the base name: a GUID that is textually represented in a way that can form a domain name label

Thus, if one wants to think of terms of domain names, a user might express a locality sensitive name of a cloud entity peer as <sub-realm-GUID>.<cloud-entity-GUID>.example.tld

Persistent Relationships

It seems that in the world of clouds that network relationships last longer than at lower layer abstractions. For example, many of my own machines ,even some of my laptop machines, maintain persistent JungleDisk relationships, even through through reboots, with Amazon's S3 cloud storage product.

It seems rather clear to me that in the world of clouds that we need something that lasts longer than TCP connections and is more flexible than HTTP “cookies”.

Which brings me to what I call an “association” protocol.

An Association Protocol

At its most basic an association protocol is something that allows two network entities to maintain an ongoing communication despite lapses in underlying connectivity or changes in addresses (or even Ipv4/IPv6 use) due to location changes.

For this there seem to be at least two architectural options:

One could build a convergence layer that sits between the IP and TCP/UDP protocols layers. Such a protocol is in fact been defined and implemented. One advantage of this approach is that applications that don't exchange IP addresses can often remain unchanged.

Or one could create a protocol layer that sits above TCP. (An association layer would not be needed above UDP.) This approach has a disadvantage in that it almost certainly would affect any application that expected TCP services. However, I believe that there is a particular advantage that make this the superior architectural approach.

The advantage is this: The kind of long-term persistence that we can expect in cloud computing will tend to be structured on the basis of transnational exchanges rather than unbounded data streaming. This suggests that applications would find it useful to have a protocol service that bounds transactions in a way that both ends to the connection would be able to know that a transaction has not completed, has completed, or is uncertain. This need dovetails very nicely an association layer protocol that exists above TCP and that has to accommodate the effects of failures of TCP connections and their replacement with new TCP connections.

Much as we like to denigrate the old ISO/OSI protocol architecture as excessively heavy and fussy, it did contain some good ideas. And one of those ideas was buried in layer 5 of their protocol stack, the Session Layer, X.225 (which apparently is still not freely available.)   Of course, as in all things ISO/OSI that good idea was hidden under a mountain of excessive, and to my mind, unnecessary, complexity.

The good idea is this: A simple protocol that does only two primary jobs beyond the transmission and reception of user data (which is framed into messages.)

First, it manages the underlying TCP transport, building and rebuilding TCP transport connections as those connections fail or go idle.

Second, it allows the two endpoints to create named checkpoints in the sequence of messages to that the two endpoints can know that the data flow has proceeded at least to a certain named point (the data flow may have proceeded further.)

The two endpoints can use this second, named checkpoint, mechanism to know how far they need to back up to re-synchronize themselves.

There is a widely held view that this requires the association protocol to snapshot application data. That is not a correct view – the association protocol merely allows the endpoints to established named markers that indicate the progress of *both* entities. Any data holding and rollback is the responsibility of the endpoints.

(All of the other stuff in the OSI Session protocol – dialogs , flow control, etc seem to be unrelated to cloud computing issues, so I'm more than willing to ignore them.)

There is an ancillary benefit to an association protocol – it would provide an alternate to the three-point routing used in mobile IP.