WHITE PAPER: **MESH NETWORKS**

Geo-Replicator®
Replication in Ad-Hoc Mesh Networks

01. Introduction

In locally synchronizing data between mobile assets, ad-hoc mesh networks involving multiple point-to-point links between nodes become necessary to enable timely propagation of changes to data[1].

The characteristics of, and requirements for replication over, such networks will be:

- Short-range links, not involving connectivity outside of a local group of nodes,
- Low bandwidth, potentially unreliable links, whose up-time cannot be guaranteed and are likely to come and go, especially in military scenarios,
- Links potentially intermittent (hence particular nodes may prove uncontactable for arbitrary periods), and nodes themselves may cease to be operational for periods of time, potentially forever,
- Some links may have different bandwidth and reliability/intermittence characteristics to others (e.g. – consider a fleet of ships with command ships and more mobile pickets, or aerial assets which may roam significant distances from the 'hub' of the fleet).

Robustness and data visibility from any particular node, even when it might be disconnected at any particular point in time, both call for data to be replicated at all nodes of the system. Hence any replication system must flow data changes from whichever node they arise at to all other nodes in a timely fashion. There are broadly two ways to achieve this:

1. By each node operating independently and broadcasting changes (both received and originating there) to all other nodes that have not yet seen said change.
2. By imposition of a routing topology (potentially dynamic since any particular route may drop out intermittently, or permanently).

The first option naively offers the benefits of simplicity and self-evident robustness. The second appears more complex, but can significantly reduce network utilization. To see why we will consider the (logical[2]) message interchanges required in each case.

[1] (see for example the discussion of subnet relays at http://findarticles.com/p/articles/mi_m0OBA/is_1_25/ai_n27149021/)

[2] This discussion is written from a protocol and product independent perspective

02. Case 1 – Point-to-Point

A point-to-point network is defined by the following statements:

- Any node originating a change broadcasts to all other reachable nodes, maintaining a list of those it has sent the data to. Note that such a broadcast must be acknowledged for the originator to be sure the data has arrived, implying a round trip to each other node in the network. It must retain a record of the specific data change (or batch of changes more likely for efficiency purposes – perhaps all changes within some fixed period) until it knows it has been received by all other nodes in the system.
- Any node in receipt of a change must forward it to other nodes that have not seen it, since the originator cannot be relied upon to remain active or to retain connection to other nodes. This means that a receiving node must also maintain a list other nodes that have seen each change just as the originator must (essentially this means that there is no difference between originating a change and receiving one from another node)
- When a previously disconnected node becomes connectable, a node must forward any changes to it that it does not know the newly connected node already has. This requires either a blind forwarding or a synchronous message exchange to query the newly connected node to determine whether or not it as seen the change before sending it. The former approach is readily ruled out since it would result in a change reflecting around the system indefinitely unless all nodes were connected at once. The latter approach requires synchronous communications between nodes, and a protocol to synchronize the list of nodes known to have any given change (without which if node A has a change and sends it to B and C, B and C will then not realize that each other already has it until they are able to speak to one another)

What does this mean for the message flows that are generated? Consider a three node network A, B, C and a change originating at A. Suppose connections are intermittent and a reliable connection of all nodes to all other nodes is not the case. For the sake of argument let's suppose node A is first able to connect to B, and then later to C. It forwards the change to B, then later to C, at which point it knows that all nodes have seen the change, and so node A can then stop trying to further propagate it. At this point the system is synchronized, but nodes B and C do not know this. When B connects to C it must query C with some protocol to determine that C has already seen the change, after which it may conclude the entire network has it and thus cease trying to propagate it. Similarly C must make a symmetrical communication with B.

02. Case 1 – Point-to-Point (continued)

The situation gets incrementally worse if we add another node, D into the mix. In this case suppose B connects to D at some point after receiving the change from originator A and communicating with C. After a synchronous query it determines that D does not have the change so it provides that change to D. However, D won't know that C also has the change, and so on next connection to C, D will again instantiate the query protocol. To avoid this it is necessary to not only pass the change to onward nodes, but also to pass the list of known recipients, and thus prevent spurious extra communications. In reality any sensible protocol would pass this list as the response to the have-you-seen-this-change query, but it becomes quickly obvious that what first appeared to be a very simple broadcast of changes, becomes a fairly involved protocol in this scheme.

Assuming a sufficiently complex scheme is adopted to synchronize recipient lists as discussed above then the total message traffic will be as follows:

- The data for the change itself will flow once to each node
- The route by which data flows will be arbitrary and may or may not map well to the bandwidth topology of the network (i.e. – changes may not route along the highest bandwidth links)
- Further synchronous exchanges are needed amongst the nodes. The exact number required will depend on the exact timing of connections:

  In an N-node network the best case is where all nodes are initially connected, in which case the originator can connect to each in turn, passing it the data and also the list of the nodes it knows have already received it. If data is sent synchronously (probably a bad idea since it means sending to different nodes one at a time, rather than in parallel) then as each node is synchronized it can also be told about the previously connected nodes and thus know that they have already received the change. This means the first node to receive the change will still need to speak to N-2 others later to determine that it does not need to forward to them. The second to N-3, and so on. This leads to a total message exchange of N-1 + N-2 + N-3 + … = N(N-1)/2. Realistically we will almost certainly want to prioritize sending the data promptly, and thus initiate parallel transfers on the available links, which means that no receiving node can be told about completed sends to other nodes at the same time. Hence each receiving node must send to all others apart from the originator, resulting in $(N-1)^2$ total exchanges.

02. Case 1 – Point-to-Point (continued)

- The worst case occurs when the pattern of first available connections forms a chain, such that the originator can only speak to one other node[3], which can in turn speak to one other and so on. In this case the data flows once as before, but takes much longer to propagate around the system. There is no saving on the overhead of the have-you-seen-this protocol - the originator eventually has to speak to every node. The first link on the chain only knows that the originator and the next link have synchronized, and so has to speak (eventually) to every other node and so on. This gives us a total of N(N-1)/2 exchanges again.

It is also the case that it is hard to ensure a convergent final state in such an ad-hoc mesh. Clearly since nodes disconnect and reconnect all the time, absolute synchronization is not typically possible until the system has been quiescent for some time. However a desirable property is convergent consistency, by which we mean that the system always moves from a less convergent state to a more convergent one with each individual node synchronization. With an ad-hoc mesh this is very hard to achieve and predict, especially when the typical change period (of any particular data item) is of the same of the same order as the total propagation time across the mesh. In such cases updates may be received out of order due to being routed differently, and additional complexity results trying to determine how to apply updates in the correct order (or ignore old ones).

For example consider a 4-node network, A, B, C, D wherein a data item is changing at A fairly frequently. Let's suppose A makes a first change a' and is then able to establish a connection to B, synchronizing that change to B. Later it makes change a'' but now happens to have a connection to C, which receives the latest data, a''. C then connects to D and passes a'' along. Later B establishes a connection to D and forwards a' (now out of sequence from D's perspective). Additional protocol complexity will be required for D to correctly resolve its final state. On a larger scale, as the mesh is partially partitioned different (and inconsistent) updates may wind up 'chasing each other' around the network over different routes, and the entire system can be very hard to stabilize to a convergent end state.

This example also illustrates difficulty in handling clashes in an ad-hoc mesh synchronization – i.e. – two conflicting data changes that originate from different sources. Since different nodes may receive each of the two conflicting changes over different links arbitrarily, there can be no central point of resolution of the 'winner'. Consequently every node must always resolve the conflict itself, and in order for the result to be convergent every node must make the same decision.

[3] Actually it's worse than this. Ideally each node speaks to each other, but if the mesh is such that some nodes never make direct connections with one another, then the result will be that the maintaining of the recipients lists for the changes becomes an ever-increasing overhead, requiring a significantly more complex protocol to forward recipient lists around the system almost as first class data to avoid eventual resource starvation.

02. Case 1 – Point-to-Point (continued)

The only information nodes can possibly have to make this choice are:

- Identity of the originating node (this could be passed around with the data)
- The data itself (including potentially a timestamp for when the originating change was made, though that relies upon exact clock synchronization between all nodes which is not always easy to achieve)

As a consequence the likely resolution mechanisms are going to be one of:

i. Based on some priority ordering of nodes so that originator priorities can be compared. This is extremely against the general philosophy of ad-hoc mesh synchronization since it effectively establishes a topology (though not a routing topology)

ii. Based on origination timestamp (latest wins probably). This has the disadvantages that the winning update will tend to appear somewhat arbitrary from the originator's viewpoint (e.g. – when the admiral's update is trounced by an ensign's), and that unless clock synchronization is maintained at all nodes fairly precisely, the system will never guarantee to converge (since any 'rogue' node with a drifting clock may resolve the same conflict differently to other nodes and propagate it's winning version preferentially)

iii. Based on some examination of the data. Apart from schemes that simply work from hashes of the data (and thus, although deterministic and therefore convergent, will appear utterly random from a user perspective), this requires the synchronization system to have built-in semantic knowledge of the data which greatly adds complexity.

03. Summary - Point to Point

Thus in summary, use of point-point-links in a pure peer fashion leads to:

- Data flowing along sub-optimal routes (not routing preferentially via high-bandwidth routes where available).
- An overhead synchronization protocol to determine which nodes require each change, involving $O(N^2)$ synchronous protocol exchanges for each data change (or distinct batch of changes).

04. Case 2 – Synchronization via an imposed (dynamic) routing topology

The alternative is to use the underlying mesh network merely as a transport for a higher-level synchronization topology. The simplest example is to impose a tree structure. To ensure connectivity any spanning tree will suffice, though intermittence of connectivity (and of node lifetime even) implies that this spanning tree may need to be adjusted dynamically in response to node or connectivity loss. Data would flow only up and down links in this spanning tree.

As an example consider a small naval taskforce consisting of:

A. *Command ship*

B. *Fast picket boat #1*

C. *Fast picket boat #2*

D. *Helicopter launched from A*

(A) Has the most reliable/highest bandwidth and range communications. (B) and (C) roam from (A) some distance but are usually in contact. (D) is more mobile and operationally may often drop out of reliable data communications. In this case it is probably appropriate to impose a simple one level tree (i.e. – hub) with (A) as root. This means synchronization traffic routes via the higher bandwidth, more reliable links provided by (A). In the alternative ad-hoc mesh synchronization exchanges will occur between (e.g.) the picket boats and the helicopter when they happen to be in range. If it happens that this occurs at a time when a large change (by data size) is awaiting synchronization, low bandwidth, possibly operationally crucial data links will be swamped by (likely relatively non-urgent) synchronization traffic. It may even happen that due to the higher transmission power available (and therefore signal strength and ultimately bandwidth) from (A), data traffic that could route via (A) most efficiently is sent between (B) or (C) and (D) instead, taking much longer than round tripping via (A).

04. Case 2 – Synchronization via an imposed (dynamic) routing topology (continued)

Recipients of data require no knowledge of the state of other nodes apart from their parent in the spanning tree (which they can asynchronously query for changes and to which they must send changes originating at themselves or their descendants in the spanning tree). This means:

- Spanning trees can be chosen based on intelligent assumptions about likely node connectively – e.g. – a capital ship is probably a good tree root because it has higher bandwidth more reliable communication links than (say) a mobile aerial asset. By selecting a spanning tree that takes regard of the connection quality (both bandwidth and likely lifetime) in this way we can ensure that data routes along optimal or near optimal routes.
- No additional protocol is required at the level of individual changes (or change batches) to track which nodes have received it (just a flag to say whether the parent node has seen it).
- Data can be 'pulled' asynchronously from a node's parent and immediate descendants, rather than requiring a synchronous exchange (the node data is being pulled from can be entirely passive in that exchange). This leads to better restart characteristics in the event of interruption part way through streaming a change, potentially even if the restart happens against a different node after a change in spanning tree topology.
- A higher level protocol to establish spanning trees is required, but in practice it is likely that a simple hub and spokes arrangement (or possibly 2-level tree of nested hub-and-spokes) that only changes if the hub is actually lost will be optimal in most cases. Such dramatic changes will be rare however and do not impose an overhead on normal operations.

05. Summary – Synchronization via an imposed (dynamic) routing topology

Thus in summary, imposing a higher level synchronization topology, using the mesh network as its transport results in:

- The same total bandwidth usage for data transmission as the ad-hoc peer-to-peer method, but optimized to flow over the higher bandwidth links (so lower, perhaps much lower, network utilization),
- No protocol overhead to determine which nodes need forwarding to, an O(N2) saving in protocol exchanges. Since each of these (probably) represents an IP connection negotiation and establishment, plus a short data exchange, this is potentially a significant saving in overall network utilization.

For more information on how iOra Geo-Replicator can help your organization, get in-touch via **support@iOra.com**