Consensus Node

Table of contents

1. Consensus Node

A consensus server is a server node whose function is to make authenticated decisions. There will normally be several such servers acting together. They make decisions by majority voting using what are called **Byzantine** protocols. All XLattice networks (lattices) must have at least one cluster of such consensus servers; it will be responsible for issuing node IDs for the lattice. There may be only one server in the cluster.

1.1. Use

A node needing to use a consensus server must first discover it and then request the desired service.

Discovery will take one of two forms. First, a physical address and public key for one or more of the nodes participating in the cluster might be supplied in a configuration file.

Alternatively, this information might be available in a well-known place, usually a Web site configured to respond to a CGI query with a text reply in a standard format.

In either case, the node needing a service establishes an encrypted connection to a physical interface on one of the server nodes and sends a message requesting the service. After voting, all of the servers will send digitally signed replies over secure connections, establishing the connection first if necessary. The client is responsible for saving the authenticated replies, determining what the consensus decision is, and acting on that consensus.

It may be that it is impossible for a bootstrapping client node to discover a consensus server. In this case the client node creates a consensus node, waits for it to bootstrap, and then requests the service from it.

1.2. Node ID Acquisition

The most basic authenticated decision service is the provision of a node ID. This is a 160-bit binary/20 byte number generated by the servers using a quasi-random process and agreed upon using a Byzantine protocol. While the node ID is not actually guaranteed to be unique, it is exceedingly unlikely that two identical node IDs would be generated in many lifetimes of the universe. Until it has obtained this authenticated ID, a node cannot communicate with other nodes.

The request for a node ID includes the Address of the client node and salt. The Address is a data structure including either a number of (one or more) physical addresses (such as an IP address/port number pair) or a node ID. In this context salt means a random 32-bit/4 byte number. If the request is repeated the same salt must be used.

Local policy will determine whether a node ID will be issued in response to a request. A default policy might be that no more than 8 node IDs will be issued for a given IP address in a

period of time, that no more than 24 node IDs be issued for a /24 over some period, and that no more than 1024 node IDs be issued for a /16 over that period.

IPv4 addresses are 32-bit numbers usually written in "dotted quad" format, a dot-separated set of four positive integers, each of which is less than 256. Examples would be 1.2.3.4 and 251.252.253.254.

A /24 is a block of 256 IP addresses whose first 24 bits are the same; a /16 is a block of 64K IP addresses whose first 16 bits are the same.

1.3. Byzantine Protocol

Byzantine protocols are used to arrive at decisions in environments where communications links and network components may be faulty or even controlled by hostile parties. The name comes from the so-called "Byzantine generals problem":

A general of Byzantium and his lieutenants are besieging a city. They communicate using signed messages. If they are to be successful, they must all attack simultaneously. If one attacks and the others hold back, the attacker will be destroyed.
Any of the parties involved may be corrupt, as might the messengers, so they need a protocol which will allow them to determine whether an attack order is authentic or whether one or more parties might be lying. They will attack only if the order is certain to be genuine.

There are many variations of the protocol solving this problem. Most involve using cryptographically authenticated messages and a number of rounds of communication (typically around N^2, where N is the number of participants). So long as more than two-thirds of the participants are honest, the problem can be solved.

In this case, a client submits a request to any of the servers. After a number of rounds of voting, all of the servers will reply. Each reply will include the number of servers. If there is a consensus, the client stores the appropriate number of authenticated replies and uses the result. If there is no consensus, or if the request times out, the client repeats the request.

The decision protocol is guaranteed to be idempotent, meaning that the same request will always return the same result or be rejected.

1.4. Trust

Consensus decisions of clusters of servers using Byzantine protocols is the basis of trust in XLattice. It should be obvious that trust is greater if

- the network is longer established
- network policies are known to be sensible
- the number of servers is greater
- individual servers have been up for longer times
- server operators are known and respected

- decisions are more often unanimous

Policies are specific to a network and can be made quite rigorous or extremely lax.

Rigorous policies might involve digital certificates from major certificate authorities, checks of organizational personnel records, photo ID checks, payment by credit card, or IP address checks, or some combination of these -- or just allowing membership to people you know face to face.

A lax policy would be to let just anyone in.

It is likely that in order for networks to grow and last policies need to be at least moderately rigorous. It would be sensible for a network to post its policies in a well-known place and to take great care in ensuring that the posted policies are the actual policy in use.

1.5. Bootstrap

When a node is initially booted, it may have no node ID in its configuration file. If it cannot discover a server from which to get a node ID, it creates a second XLattice node, which becomes a single-node XLattice consensus server. This will issue itself a node ID and then accept a request from the bootstrapping node for a node ID. Once this has been issued, the lattice will consist of two nodes: the original client node and the consensus node, both running on the same platform.

A solitary consensus node simply makes decisions. It does not use any Byzantine voting process because there are no other nodes to reach agreement with.

1.6. Joining and Leaving

Consensus nodes operate as a separate p2p network, a lattice. All nodes in the lattice are permanently linked by a full mesh of encrypted communications channels. Each server sends a keep-alive message including a member count to all of its peers at a configurable interval, normally 60s.

If a node does not receive a keep-alive from a peer within this period, it requests one. If that request times out, the node can propose that the non-replying node be dropped from the server lattice. The nodes vote on this proposal using the usual Byzantine protocol. If the proposal succeeds, the failing node is dropped from the lattice and the number of members decreased by one.

A node may apply to join the consenus lattice. To do so it must already have a node ID. It sends a join request to any member of the lattice. That member appends its decision, signs the request, and copies it to all other lattice nodes. After the usual Byzantine rounds of voting, if the new member is provisonally accepted, all members establish encrypted links with the new member and send it a keep-alive. If that is replied to, the new node is a member of the lattice.

1.7. Fragmentation and Merger

Under certain conditions the server lattice will fragment, leaving two or more subsets of nodes internally connected but with no links between members of different subsets. This might occur, for example, if a corporate LAN loses its Internet connectivity. In such a case nodes on the corporate LAN would constitute one well-formed lattice and those elsewhere in the Internet another.

Under other conditions, two or more well-formed lattices will want to merge. This will occur in the case above when the corporate LAN regains its Internet connectivity.

We do not at this time have a sketch of the protocol for handling these two conditions, but it will be necessary to develop such.

1.8. Performance Considerations

In certain applications it will be important to authenticate decisions but equally important that this be done quickly. This will be true in both interactive games and messaging systems, for example. The key factor here is that making decisions involves passing messages, with the number of messages O(N^2), on the order of the square of the number of servers. Under these circumstances the number of servers should be small, bearing in mind that if there are less than four servers all decisions will have to be unanimous.