The Design of Bionet 2

Kevin Gifford, Sebastian Kuzminsky, Shea Williams

Version 1.0 preliminary

2007 December 11

Table of Contents

Illustration Index

1 Introduction

Bionet is a framework for distributed, soft-realtime sensing and control. Bionet uses a secure, scalable, pub/sub based peer-to-peer communication system. Bionet is designed to handle disparate end-point devices in a standard way, to support device mobility, and to work well on intermittently connected networks.

1.1 Terminology conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2 Overview

Bionet is a peer-to-peer system. There are two kinds of peers in a Bionet network: Hardware Abstractors (HABs) and Clients.

HABs act as device drivers, interfacing different sensor and actuator devices to Bionet.

Clients read the data that HABs produce, and produce actuator setting commands for the HABs.

Generally, HABs communicate with Clients but not with other HABs, and Clients communicate with HABs but not with other Clients.

HABs and Clients communicate over an IP network using a combination of unicast and multicast.

Bionet is designed to deal gracefully with network tear-away events, defined as the partitioning of one network into disjoint components. After a network tear-away, each network fragment continues to work as fully as possible. After a network reconnect, Bionet synchronizes and reconnects all components. We assume that network reconnect events join a torn-away network back to its original place in the network topology; ie IP addresses and routing tables do not need to change to effect network reconnects.

This illustration shows all components of Bionet and their relationships:

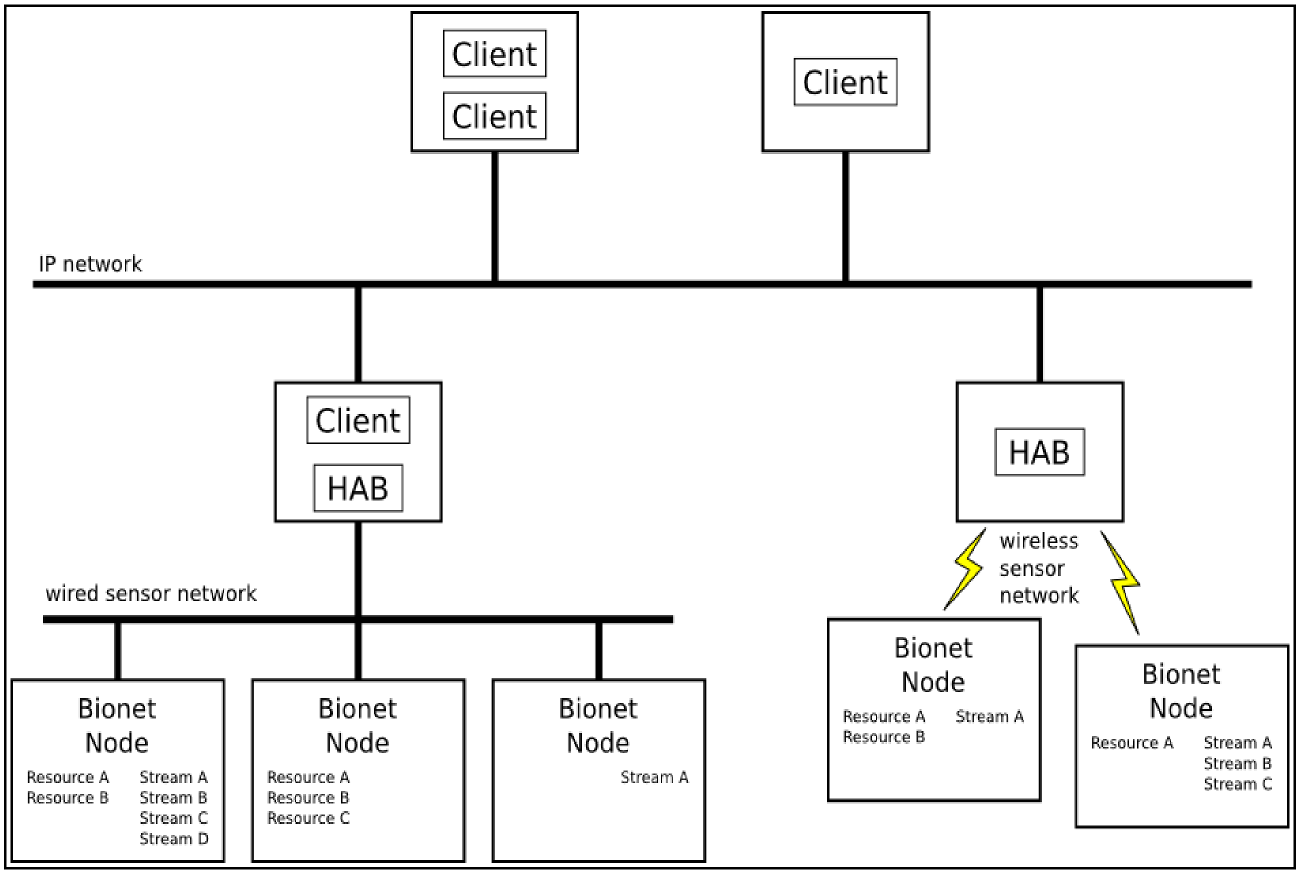

Illustration 1: Bionet Network Overview

An IP network connects a number of Clients and a number of HABs executing on various computers. The HABs act as gateways to various sensor networks. Devices resident on the HABs' sensor networks are represented as Bionet Nodes, and their various I/O capabilities are represented as Bionet Resources and Streams. (We speak of "sensor networks", but end-point devices may include actuators and effectors as well as sensors.)

HABs and Clients are processes executing on computers; Nodes, Resources, and Streams are software objects that represent real devices.

One thing that's not show in the figure is the dynamic nature of Bionet. Clients and HABs can come and go at any time. Nodes can come and go at any time. There is no a priori knowledge of what Nodes are available, or what their internal structure is.

2.1 Decentralized design

There are levels of decentralization. The less a network relies on central servers, the more decentralized it is. Some P2P systems use centralized servers for membership management, directory services, Access Control Lists (ACLs), or other things.

We have tried to design Bionet to operate with minimal reliance on central servers. Bionet requires a trusted ACL server (to enable peers to authorize each other's requests), and the AMS-based protocol requires some additional servers to supervise communications.

2.2 Security

Bionet has a fledgeling security model that, together with lower-layer network security mechanisms such as IPsec, addresses four aspects of network security:

- Authentication
- Authorization
- Confidentiality
- Integrity

Bionet makes use of one piece of trusted infrastructure: an ACL server (required for authorization).

2.2.1 Authentication

Authentication is handled by public key cryptography. Each Bionet process has a unique key pair; the public key is used as a claim of identity and the private key is used as proof of identity.

Bionet Clients that that are user interfaces generally operate on behalf of a human rather than on behalf of themselves. In such cases the identifying key pair SHOULD correspond to the human operator rather than the UI program. When the operator starts the program they provide it with their key pair.

Bionet HABs require a cryptographically strong mapping between their HAB Name and their public key. This is provided by a certificate signed by a trusted certificate authority.

2.2.2 Authorization

Authorization is handled by Access Control Lists.

A trusted ACL server tracks which processes (identified by public keys) are allowed to perform which operations. When a Client makes a request of a HAB, the HAB submits a query to the ACL server of the form “is the Client identified by <public key> on the Access Control List <ACL ID>?”. The ACL server replies to the HAB with a signed certificate either confirming or denying access, and the HAB uses this information to allow or forbid the Client to complete the requested operation.

Bionet provides fine-grained access control. Separate ACLs are used for:

- Subscribing to each HAB's Node List
- Subscribing to each Resource
- Setting each (writable) Resource
- Reading each (readable) Stream
- Writing each (writable) Stream

2.2.3 Confidentiality

Confidentiality is used to protect sensitive information such as personal medical data from unauthorized access by insiders and eavesdroppers.

The bulk of Bionet's confidentiality needs is handled by symmetric key encryption.

The remainder of Bionet's confidentiality needs is to handle the distribution of symmetric keys, and a few other occasional, low-rate transmissions; this is handled by public key encryption.

Bionet uses one symmetric key per subscription topic. When a process requests to subscribe to a topic, the publisher authenticates the requester's identity (using public key cryptography), then verifies authorization by asking the ACL server. If both these checks pass, the publisher sends the requester the symmetric key for the topic (encrypted with the requester's public key).

2.2.4 Integrity

Integrity is handled by digital signatures where appropriate, and by message integrity codes or checksums.

2.2.5 Security Problems

The Bionet protocols as described in this document notably do not protect against replay attacks. For example, an adversary that can sniff and inject packets can capture HAB Join and HAB Leave messages and replay them later, causing Clients to believe that the HAB has come or gone.

There are probably additional attacks that can be effective against a Bionet network.

Rather than try to solve all security problems within Bionet, we have chosen to rely on external security measures below Bionet in the protocol stack to hopefully close these holes. Many lower-layer security mechanisms are potentially useful depending on what threats the system designed wants to protect against. IPsec and 802.11i are probably the most applicable.

The NASA C3I specification includes IPsec (see NASA Constellation Program (CxP), "Constellation Program Command, Control, Communication, and Information (C3I) Interoperability Standards Book, Volume 1 - Interoperability Specification, CxP 70022-01, Section 3.4.1 Network Layer Security, Feb. 12, 2007).

We have examined some IPsec use cases below for reference.

2.2.6 IPsec

Shea will write this. IPsec doesnt appear to work with multicast. Doh. FIXME

2.3 Data structures

Bionet peers communicate about four kinds of data objects: HABs, Nodes, Resources, and Streams.

This section presents the Bionet data structures. It starts with the simplest structures and builds up from there. Use of the data structures and meaning of the fields is covered in the following section.

2.3.1 Resource structure

Each Resource has the following attributes:

- Resource ID: A string consisting of one or more characters from the set [-a-zA-Z0-9] (the dash character “-”, lowercase letters, uppercase letters, and numbers). The ID of a Resource must be unique among all the Resources of its Node.
- Access Mode: Either Read-Only or Read-Write. This defines what operations are allowed on this Resource.
- Data Type: One of floating point (single or double precision), integer (signed or unsigned; 8, 16, or 32 bits), or binary. New Data Types can easily be added.
- Value: The most recent value (of the appropriate Data Type).
- Timestamp: The time (with microsecond resolution) that the most recent value was acquired.

The Resource ID, Access Mode, and Data Type are static metadata, they never change after the Resource has been defined. The Value and Timestamp are time-varying data.

2.3.2 Stream structure

Each Stream has the following attributes:

- Stream ID: A string consisting of one or more characters from the set [-a-zA-Z0-9]. The ID of a Stream must be unique among all the Streams of its Node.
- Direction: Producer or Consumer.
- MIME Type: Internet media type of the stream data. See

http://www.iana.org/assignments/media-types/

All attributes of a Stream are static metadata, they never change after the Stream has been defined.

2.3.3 Node structure

Each Node has the following attributes:

- Node ID: A string consisting of one or more characters from the set [-a-zA-Z0-9]. The ID of a Node must be unique among all the Nodes of its HAB.
- Resource List: A possibly empty list of Resource objects.
- Stream List: A possibly empty list of Stream objects.
- Fingerprint: A hash of all the static metadata of the Node (ie, everything except the Values and Timestamps of the Resources). Used for Node List Subscriptions (see below).

All attributes of a Node are static metadata, they never change after the Node has been defined.

If a HAB wants to change anything about a Node, such as its Resource List or Stream List, it must remove the Node from Bionet and add the modified Node.

2.3.4 HAB structure

Each HAB has the following attributes:

- HAB Type: A string consisting of one or more characters from the set [-a-zA-Z0-9]. The Type of a HAB specifies the kind of sensor network that the HAB connects to. There may be multiple HABs that have the same HAB Type.
- HAB ID: A string consisting of one or more characters from the set [-a-zA-Z0-9]. The ID of a HAB differentiates the HAB from others of its Type. The HAB ID must be unique among all the HABs that share its HAB Type.
- Delivery Vector: A non-empty list of unicast addresses for communicating with the HAB.
- Node List: A possibly empty list of Node objects.
- Node List Fingerprint: A possibly empty list of Node Fingerprints, one for each Node in the Node List. Used for Node List Subscriptions (see below).

The HAB Type, HAB ID, and Delivery Vector are static metadata, they never change after the HAB has joined the Bionet network. The Node List and Node List Fingerprint are dynamic, they change as Nodes join and leave the HAB's sensor network at runtime.

NOTE: The Node List Fingerprint is designed to be simple. If it turns out to be too inefficient with large Node Lists, we may switch to a Merkle tree for the Node List Fingerprint.

HAB objects are unique among the Bionet objects in that they correspond to processes; peers on the Bionet network. A HAB is both a process executing on a computer and a software representation of that process in the Bionet hierarchy of objects.

2.4 Ontology

Bionet organizes its data objects into a hierarchy outlined in the previous section. This abstraction is not universally applicable, but it has served well as a workable model of a wide range of actual sensor networks.

The hierarchy is reflected in the Bionet naming schema, which is used to refer to objects on the network.

2.4.1 HABs

HABs are uniquely identified by the combination of their HAB-Type and HAB-ID.

The HAB-Type indicates the kind of sensor network the HAB interfaces with. There may be multiple separate instances of a particular kind of end-point device or end-point device network, so HAB-Type is not necessarily unique. Example HAB-Types are:

- ALSA: Interfaces to audio devices (microphones and speakers, and their related volume and equalizer controls) via the Advanced Linux Sound Architecture (ALSA)
- CSA-CP: Interfaces via RS-232 to a single CSA-CP device
- Alien-RFID: Interfaces via TCP/IP to an Alien Technology ALR-9800 RFID reader

The HAB-ID differentiates among all the different HABs of a particular HAB-Type. The HAB-ID is chosen when the network is deployed, and should generally be evocative of the location of the end-point hardware; this provides coarse-grained localization of end-point hardware.

From the HAB-Type and HAB-ID we construct the HAB Name, which is the HAB-Type and HAB-ID joined by a dot (“.”).

For example, a deployment of Bionet might include two ALR-9800 RFID readers in different rooms (let's call the rooms “hangar” and “storage”). The Bionet network would include two instances of the Alien-RFID HAB to interface to the two readers. The two HABs would share the HAB-Type Alien-RFID, but they would have different HAB-IDs, probably identifying the rooms housing the actual RFID reader hardware. The HAB Names would be “Alien-RFID.hangar” and “Alien-RFID.storage”.

To build on this example, the network might include intercoms in the hangar and the storage room. The intercoms might be controlled by two ALSA HABs named “ALSA.hangar” and “ALSA.storage”.

2.4.2 Nodes

A Node in Bionet is an end-point device on a particular sensor network. The sensor network that a Node resides on is identified by the HAB Name of the HAB that presents the Node to the Bionet network.

Within a particular sensor network (ie, within a particular HAB), Nodes are uniquely identified by their Node ID. The HAB that presents the Node to Bionet chooses the Node ID. The Node ID is generally derived from the end-point device's MAC address or similar: some unique, innate identifier.

Just as HAB Names are globally unique HAB identifiers made up of “HAB-Type.HAB-ID”, so Node Names are globally unique Node identifiers made up of “HAB-Type.HAB-ID.Node-ID”.

Nodes are modelled as potentially complex devices with multiple sensors, effectors, and internal state variables.

Continuing the example from the previous section, the intercom in the hangar might have two fixed audio stations, both connected to the intercom machine via USB. One might be mounted at the airlock and one at the interior door. The ALSA.hangar HAB might represent these to Bionet as “ALSA.hangar.0” and “ALSA.hangar.1”, based on the USB interfaces the audio hardware connects through.

For the RFID part of the example, a shipment of goods marked with RFID tags might arrive at the hangar. The Alien-RFID.hangar HAB would discover the tags and report each as a Node. The RFID tags have unique 96-bit IDs, and the Alien-RFID HAB naturally uses the tag IDs for the Node IDs. So the Alien-RFID.hangar hab would report a bunch of Nodes with Node Names like “Alien-RFID.hangar.0011-2233-4455-6677-8899-AABB”. As the goods were moved from the hangar to the storage room, the Alien-RFID.hangar.0011-2233-4455-6677-8899-AABB Node would be lost, and a new node, Alien-RFID.storage.0011-2233-4455-6677-8899-AABB, would be added.

2.4.3 Resource and Stream structures

Resources represent sensors, effectors, and state variables of a Node. Streams also represent sensors and effectors. Resources and Streams differ primarily in that Resources have individual, timestamped values, whereas Streams do not have timestamped values. Resources can be thought of as datagram-like, and Streams can be thought of as, well, stream-like. End-point hardware is represented by Resources when the data is of a generally discrete nature, such as readings from a thermometer or settings of a relay. End-point hardware is represented by Streams when the data is of a generally continuous nature, such as an audio stream from a microphone, or a video stream from a camera.

Ultimately it's a judgement call how a HAB should represent the information to Bionet.

Each Resource has an Access Mode, which can be either Read-Only or Read-Write. Read-Only Resource are used to represent sensors like thermometers, pressure sensors, etc. Read-Write Resources are used to represent both effectors (such as relays and analog outputs, etc) and state variables (such as sample frequency and filter decay rate, etc).

Each Stream has a Direction, which can be either Producer or Consumer. Producer Streams make information available to subscribers on the network; Consumer Streams accept information from the network.

An example of a Stream Producer would be a video camera: it makes available a video to Clients on the network. An example of a Stream Consumer would be a speaker: it accepts information from the network and makes the sound come out of the speaker.

Resources and Streams have Names much like HABs and Nodes, but the Resource ID is separated from the Node Name by a colon (“:”) instead of a dot (“.”). (This is so that a future version of Bionet might allow HABs to export Resources without the risk of confusing Nodes and Resources.)

Continuing the example runs in to a slight problem: the Alien RFID tags export no Streams or Resources. The ALSA intercom Nodes, however, export both. Typical intercom hardware will have a microphone with a mute control, and a speaker with volume control. The ALSA HAB might represent the microphone and speaker as a pair of Streams, and the mute and volume controllers as Resources. The Stream Names might be: “ALSA.hangar.0:Speaker” and “ALSA.hangar.0:Microphone”. The Resource Names might be “ALSA.hangar.0:Speaker-volume” and “ALSA.hangar.0:Microphone-mute”. You can not tell by examining a Stream Name or Resource Name whether they refer to a Stream or a Resource, but the objects have different data types internally and software cannot confuse them.

2.5 Network Conversations

Bionet peers engage in five kinds of conversations: HAB List subscriptions, Node List subscriptions, Resource subscriptions, Resource Setting requests, and Stream subscriptions.

2.5.1 HAB List

As noted above, HABs are identified by globally unique HAB Names. HABs can start and stop at any time, resulting in two kinds of events: New HAB and Lost HAB, respectively.

All Bionet Clients are always automatically, implicitly subscribed to the HAB List topic, meaning that they receive notification every time a HAB starts or stops. This information is the only in Bionet not restricted by ACLs or protected by encryption. Bionet makes no secret of what HABs are available.

The New HAB event carries with it enough information for Clients to communicate with the HAB. All subsequent Client to HAB communications is with a specific HAB identified by this implicit HAB List subscription.

2.5.2 Node List

Nodes are identified by globally unique Node Names. Nodes are software objects presented to the Bionet network by HABs. Each Node belongs to a specific HAB. Nodes can come and go at any time, resulting in two kinds of events: New Node and Lost Node.

The New Node event carries with it a complete description of the Node, including its Resource and Stream lists.

Once a Client knows of a HAB (via the implicit HAB List subscription described above), it can request to subscribe to that HAB's Node List. The Client sends a Node List subscription request, which the HAB may accept or reject.

If the HAB accepts the Node List subscription request, then the Client is informed of all the HAB's current Nodes, and of all future New Node and Lost Node events.

2.5.3 Resource

Resources are identified by globally unique Resource Names. Each Resource belongs to a specific Node. The value of a Resource may change at any time, resulting in a Resource Update event.

The Resource Update event carries with it the new Resource Value and Resource Timestamp.

Once a Client knows of a Node (via a Node List subscription), it can request (of the Resource's Node's HAB) to subscribe to the value of the Resource. The Client sends a Resource subscription request, which the HAB may accept or reject.

If the HAB accepts the Resource subscription request, then the Client is informed of the

Resource's current value, and of all future Resource Update events.

2.5.4 Resource Settings

Some Resources have Read-Write Access Mode, meaning that Clients can command their value. This is done by sending a Resource setting request to the Resource's Node's HAB. The HAB may accept or reject the request. If the request is accepted, the HAB will cause the end-point hardware to change to reflect the command; once this is accomplished the HAB will emit a Resource Update event as described above.

2.5.5 Streams

A Stream can be either a Producer or a Consumer of data.

2.5.5.1 Producer Streams

Clients can connect to Producer Streams to receive the data of the Stream. Multiple Clients can be receiveing data from the same Producer Stream, just as multiple Clients can be receiving Resource Update events from the same Resource.

For example, a HAB might export a Node representing a video camera. The video camera Node might export a Stream Producer called “video”. Multiple Clients could connect to this Stream and all view the video.

2.5.5.2 Consumer Streams

Clients can connect to Consumer Streams to provide data for the Stream. If multiple Clients are providing data for the same Consumer Stream, the behavior is determined by the HAB, just as arbitration between multiple Resource Setting commands is.

For example, a Consumer Stream representing a speaker might sensibly have several sources of audio data mixed together by the HAB and played out simultaneously.

2.6 Bionet Communication Modes

Clients and HABs communicate using a combination of multicast and unicast. The Client may choose which of these communications modes each conversation will use. The only exception is the HAB List conversation, which only runs over multicast.

Multicast is more efficient but less reliable; unicast is less efficient but more reliable.

In multicast mode there is one multicast group per subscription topic. Subscribing in multicast mode involves finding and joining the appropriate multicast group. Publishing in multicast mode involves sending a single copy to the multicast group; all subscribers receive the same message. Multicast messages use unreliable datagram transports, and some effort has gone into making multicast communications more reliable.

In unicast mode there is one connection per subscriber-publisher pair. All subscribed-to publications are multiplexed onto this connection. Publishing in unicast mode involves sending the item to each subscriber individually.

3 HAB List and peer discovery

Peer discovery and HAB List subscriptions are accomplished by the same mechanism. All Clients are implicitly subscribed to Bionet's HAB List topic (though whether the HAB List events stay in the Bionet library or reach the application is a choice left to the application author).

The HAB List/peer discovery protocol uses a combination of multicast and unicast. All multicast traffic uses UDP on a well-known multicast address and port.

FIXME: or maybe we'll just use mDNS/DNS-SD instead

3.1 HAB responsibilities

3.1.1 HAB List Subscription: HAB state diagram

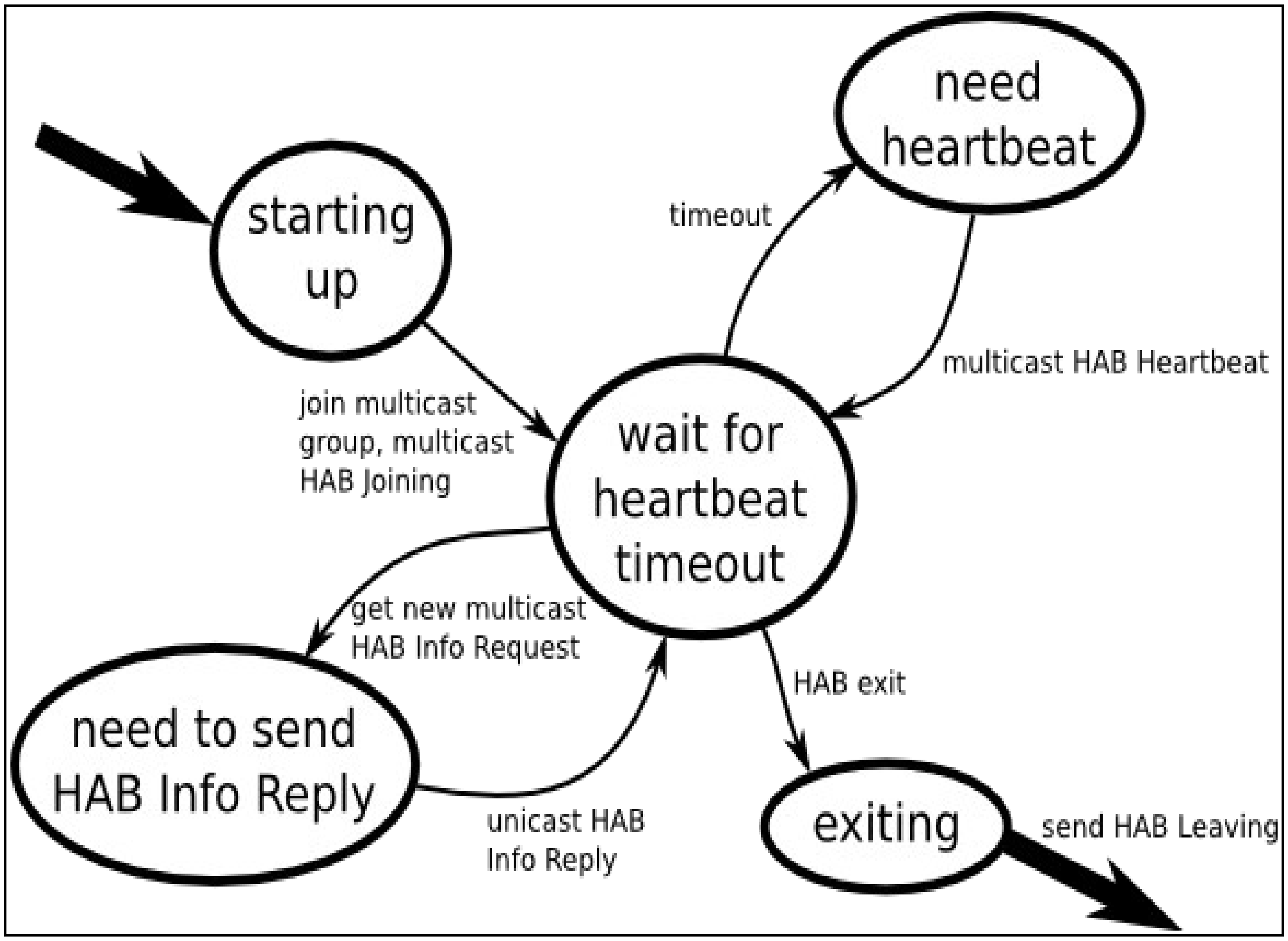
starting up
need heartbeat
timeout
multicast HAB Heartbeat
join multicast group, multicast HAB Joining
wait for heartbeat timeout
get new multicast HAB Info Request
HAB exit
need to send HAB Info Reply
unicast HAB Info Reply
exiting
send HAB Leaving

Illustration 2: HAB List Subscription: HAB state diagram

3.1.2 HAB Joining

When a HAB starts up, it MUST join the Bionet HAB List group. The HAB MUST use the SO_REUSEADDR socket option to enable multiple processes on a single host to receive copies of the multicast traffic.

It MUST then announce itself to the network by multicasting a "HAB Joining" packet on the HAB List channel. The HAB Joining packet looks like this:

```
Message-Type: HAB Joining
Delivery-Vector: <list of HAB's unicast endpoints>
Join-Time: <timestamp>
Identity-Certificate:
      HAB-Name: <HAB-Type.HAB-ID>
      Public-Key: <HAB's public key>
      CA-Signature: <IC signed by CA private key>
```

The message is signed with the HAB's private key and sent to the HAB List multicast group.

This ensures that all running Clients discover new HABs quickly. The HAB public key is used to verify the signature of the message. The CA-Signature provides cryptographically strong proof of identity (mapping between HAB Name and HAB public key).

The HAB Joining packet can be lost (it goes over UDP), but hopefully a later HAB Heartbeat (see below) will make it.

Every HAB Name (“HAB-Type.HAB-ID”) SHALL be unique.

3.1.3 HAB Heartbeat

Running HABs MUST periodically multicast HAB Heartbeat messages to the HAB List group. The HAB Heartbeat message looks very similar to the HAB Joining message, only the message type differs:

```
Message-Type: HAB Heartbeat
Delivery-Vector: <list of HAB's unicast endpoints>
Join-Time: <timestamp>
Identity-Certificate:
      HAB-Name: <HAB-Type.HAB-ID>
      Public-Key: <HAB's public key>
      CA-Signature: <IC signed by CA private key>
```

The message is signed with the HAB's private key and sent to the HAB List multicast group.

We use different messages for HAB Joining and HAB Heartbeat to help detect the condition where a HAB dies without saying goodbye, and comes back before the heartbeat timeout.

HAB Heartbeats help ensure that Clients have complete knowledge of all connected HABs, and also provide resynchronization when previously separated networks are connected.

3.1.4 HAB Leaving

When a HAB wants to leave the network, it MUST multicast a sequence of "HAB Leaving" messages. The exact retransmission pattern is TBD, but maybe something like one per second for 5 seconds.

```
    Message-Type: HAB Leaving
    HAB-Name: <HAB-Type.HAB-ID>
    Leave-Time: <timestamp>
```

The HAB Leaving message is signed with the HAB's private key and sent to the HAB List multicast group.

3.2 Client responsibilities

Every Client SHALL maintain a HAB List. The HAB List records, for every HAB known:

- the HAB's Name
- the HAB's public key
- the HAB's delivery vector
- the HAB's last-seen time

When a Client starts up, it SHALL initialize its HAB List to be empty.

3.2.1 HAB List Subscription: Client state diagram

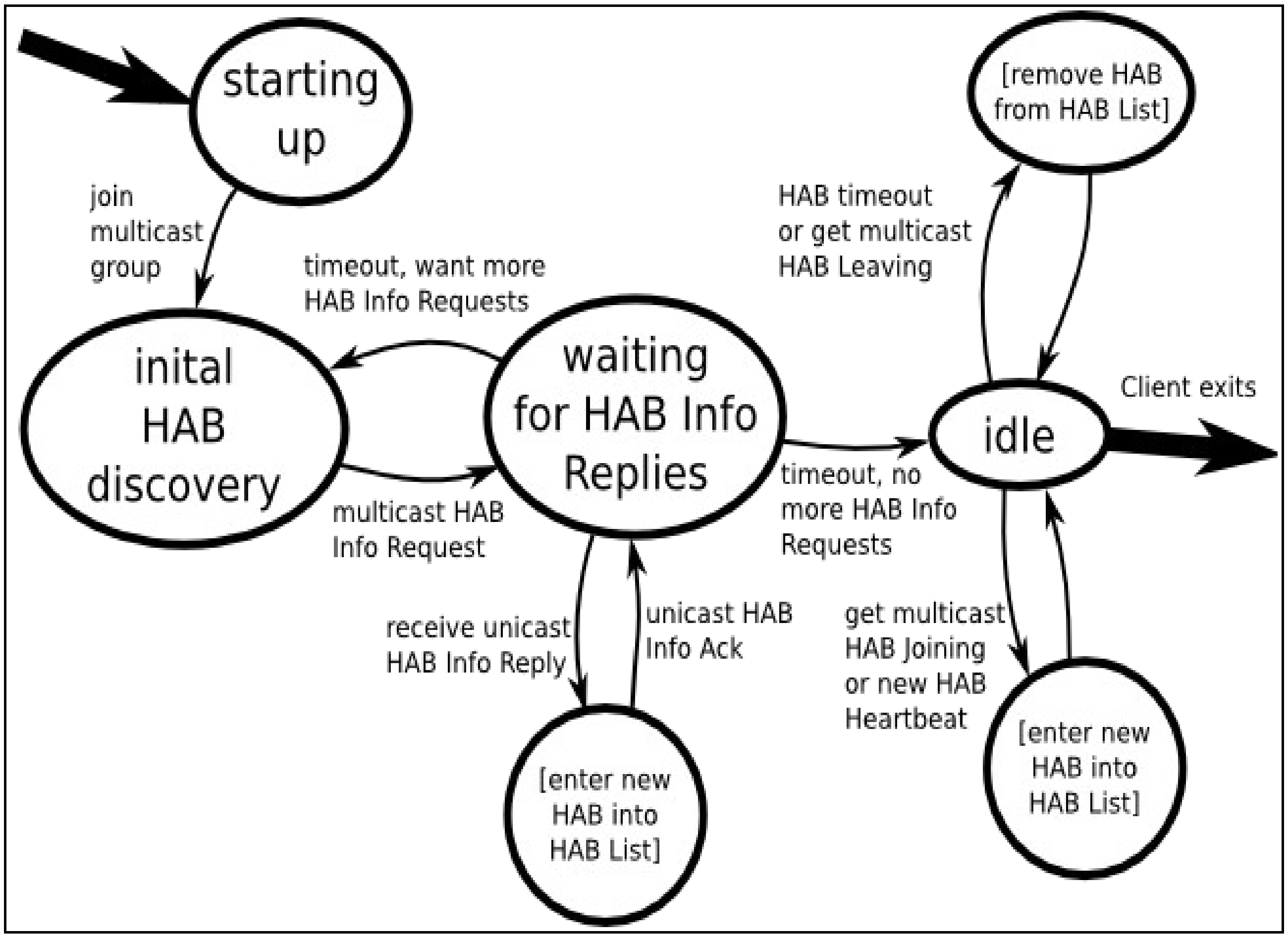
starting up
join multicast group
inital HAB discovery
timeout, want more HAB Info Requests
waiting for HAB Info Replies
multicast HAB Info Request
receive unicast HAB Info Reply
unicast HAB Info Ack
[enter new HAB into HAB List]
timeout, no more HAB Info Requests
idle
Client exits
HAB timeout or get multicast HAB Leaving
[remove HAB from HAB List]
get multicast HAB Joining or new HAB Heartbeat
[enter new HAB into HAB List]

Illustration 3: HAB List Subscription: Client state diagram

3.2.2 Client startup

At startup the Client MUST join the HAB List multicast group. The Client MUST use the SO_REUSEADDR socket option to enable multiple processes on a single host to receive copies of the multicast traffic.

The Client MUST send a HAB Info Request message to the group. The request includes:

```
Message-Type: HAB Info Request
Client-Delivery-Vector: <list of Client's unicast endpoints>
Client-Key: <Client's public key>
```

The message is signed with the Client's private key.

When a HAB receives a HAB Info Request, it MUST unicast a HAB Info Reply back to the requesting Client. The HAB Info Reply is same as the HAB Joining packet, just the Message-Type differs:

```
Message-Type: HAB Info Reply
```

```
    Delivery-Vector: <list of HAB's unicast endpoints>
    Join-Time: <timestamp>
    Identity-Certificate:
          HAB-Name: <HAB-Type.HAB-ID>
          Public-Key: <HAB's public key>
          CA-Signature: <IC signed by CA private key>
```

The HAB Info Reply message is signed with the HAB's private key.

To avoid a packet storm if there are many HABs, responding HABs wait a short, random amount of time before replying. Exact wait-times are TBD, but 0-250 ms seems reasonable. HABs SHOULD seed their RNGs with their MAC address, as per RFC 3927.

When a Client receives a HAB Info Reply, it MUST first verify the signature of the HAB with the public key given in the message, and verify the signature of the identity certificate. If the signatures are valid, the Client MUST then enter the HAB into its list of known HABs, and unicast a HAB Info Ack back to the HAB:

```
    Message-Type: HAB Info Ack
```

Upon reciept of the HAB Info Ack, the HAB registers the Client in its list of Clients replied to. If it receives HAB Info Requests from Clients that appear in this list, it silently ignores the request.

The Client SHOULD retransmit the HAB Info Request from several times, acking any replies it gets. The Client MAY repeat the HAB Info Request a set number of times, or it MAY repeat it until a few sequential HAB Info Requests bring no HAB Info Replies.

This will make reasonably sure that the Client has a complete list of the connected HABs.

3.2.3 Dealing with new HABs

If a Client receives a HAB Join packet, it SHALL look up the HAB in its HAB List.

If the HAB is not found, this indicates a new HAB has joined the network, and the Client SHALL enter the HAB into its HAB List.

If the HAB is found in the HAB List, it indicates the HAB crashed (left the network without saying goodbye) or its HAB Leaving message got lost, and it is now joining back in. The Client MUST react to this as if though a HAB Leaving message had been received before the HAB Joining message. That is, the Client should consider the HAB lost, and then found again.

If a Client receives a HAB Heartbeat packet, it SHALL look up the HAB in its HAB List.

If the HAB is found, it resets the last-seen time.

If the HAB is not found, the Client MUST react to this as if though it were a HAB Joining message.

If a HAB's last-seen time in the HAB List is older than some TBD timeout (maybe 3 times the heartbeat period), the HAB is considered lost and MUST be removed from the HAB-List.

4 Bionet over Unicast

Bionet 2 is evolved from a previous version confusingly called both Bionet 0 and Bionet 1. This earlier version used a message broker which acted as an intermediary between the HABs and Clients. All communication with the message broker used unicast.

In Bionet 2 we've taken the earlier Bionet unicast protocol and largely moved it into the HABs, removing the central message broker and switching to a peer-to-peer communications architecture. Anyone familiar with the original Bionet protocol should recognize the Bionet-over-Unicast protocol presented here.

Unicast communications go over TCP. A simple framing layer breaks the TCP stream into discrete messages.

If at any point a message fails (due to invalid signature, invalid Message Integrity Code, etc), the connection is closed by the party discovering the problem.

4.1 Initial connection

When a Client wants to subscribe to information from a HAB, it first looks the HAB up in its HAB List. This tells the Client the HAB's delivery vector and public key. The Client makes a TCP connection to the HAB and sends a Client Handshake message:

```
Message-Type: Client Handshake
Client-public-key: <Client's public key>
```

The message is signed with the Client's private key, then encrypted with the HAB's public key.

The HAB receives the message, decrypts it with its private key, and verifies that the signature matches the Client's public key. If any of this fails, it sends an error message back to the Client and closes the connection.

Assuming the Handshake is good, the HAB makes up a unique symmetric key for this connection and sends back a HAB Handshake message:

```
Message-Type: HAB Handshake
Symmetric-key: <key for symmetric cipher>
```

The message is signed with the HAB's private key, then encrypted with the Client's public key and send to the Client.

The Client decrypts the message, verifies the signature, and records the symmetric key. The symmetric key is unique to this connection, and will be used to encrypt any subsequent sensitive messages.

All subsequent messages on this connection are integrity-protected with a Message Integrity Code (MIC), then encrypted with the connection's symmetric key. A message with a bad MIC causes the receiver to close the connection.

4.2 Node List

The Client sends a Node List Subscription Request message to the HAB:

```
Message-Type: Node List Subscription Request
Node-ID-Pattern: <Node ID or wildcard>
```

The HAB checks if the Client is on its Node List ACL. If the Client is not authorized the HAB returns an error.

If the Client is authorized, the HAB records the Client's subscription, then reports all currently known Nodes which match the specified Node ID pattern (using the New Node message, one per Node), and finally returns an OK message to the Client. In the future, as Nodes join and leave the network, additional New Node messages and Lost Node messages are sent to the subscribed Client.

The OK message is simple:

```
Message-Type: OK
```

4.2.1 New Node

The New Node message fully describes the Node:

```
Message-Type: New Node
Timestamp: <timestamp>
Node-ID: <Node ID>
Resources: <list of all Node's Resources>
Streams: <list of all Node's Streams>
```

4.2.2 Lost Node

The The Lost Node message is simple:

```
Message-Type: Lost Node
Timestamp: <timestamp>
Node-ID: <Node ID>
```

4.2.3 Node List Unsubscription

When the Client wants to unsubscribe, it sends a Node List Unsubscription request:

```
Message-Type: Node List Unsubscription Request
```

The HAB stops sending Node List updates.

4.3 Resource Updates

The Client sends a Resource Subscription Request message to the HAB:

```
Message-Type: Resource Subscription Request
Node-ID-Pattern: <Node ID or wildcard>
Resource-ID-Pattern: <Resource ID or wildcard>
```

The HAB first records the Client's subscription. Then , for each Node matching the Node ID pattern, the HAB reports the current value of all Resources which match the Resource ID pattern and for which the Client is authorized to read. This reporting is done using the Resource Update message. The HAB then returns an OK message to the Client. In the future, as Resources change value and as new Nodes join, additional Resource Update messages are sent to the subscribed Client.

The Resource Update message looks like this:

```
Message-Type: Resource Update
```

```
Node-ID: <Node ID>
Resource-Updates: <list of Resource updates>
```

Each item in the Resource-Updates list specifies a Resource ID and its current Value and Timestamp.

Resources which match the Node ID and Resource ID patterns but for which the Client lacks read authorization are omitted from the message.

When the Client gets tired of the Resource Updates, it sends a Resource Unsubscription Request:

```
Message-Type: Resource Unsubscription Request
Node-ID-Pattern: <Node ID or wildcard>
Resource-ID-Pattern: <Resource ID or wildcard>
```

The Node-ID-Pattern and Resource-ID-Pattern must match an earlier Resource Subscription Request. The HAB stops sending Resource Updates.

4.4 Resource Setting

This is the only conversation which is request/reply, rather than publish/subscribe.

To command a read/write Resource to a new value, the Client sends a Set Resource message:

```
Message-Type: Set Resource
Node-ID: <Node ID>
Resource-ID: <Resource ID>
New-Value: <value>
```

The HAB verifies that the Client is authorized to write the specified Resource, then sends the command on to the Node and returns an OK message to the Client.

If the Node accepts and acts on the command, the Resource Value will change, resulting in a Resource Update being published to the Resource Update subscribers.

4.5 Streams

Streams behave differently depending on if their Direction is Producer or Consumer.

4.5.1 Producer Streams

Clients can connect to Producer Streams much like they subscribe to Resources.

When a Client wants to connect to a Producer Stream, it composes a message like this:

```
Message-Type: Stream Connection Request
Node-ID: <Node ID>
Stream-ID: <Stream ID>
```

The HAB verifies that the Client is authorized to read the Stream. If the check fails, the HAB sends back an Error message to the Client. If the check passes, the HAB composes a Producer Stream Connection Accepted message like this:

```
Message-Type: Producer Stream Connection Accepted
Stream-Delivery-Vector: <list of delivery endpoints>
Symmetric-Key: <symmetric key>
```

The Client connects to one of the delivery endpoints and reads the Stream data.

4.5.2 Consumer Streams

Clients can provide data for Consumer Streams. This is somewhat like providing Resource Settings, except continuously and generally at higher bandwidth.

When a Client wants to connect to a Consumer Stream, it composes a message like this:

```
Message-Type: Stream Connection Request
Node-ID: <Node ID>
Stream-ID: <Stream ID>
```

Note that the Stream Connection Requests for Producer and Consumer Streams are identical.

The HAB checks the Client's public key against the Stream's write-permission ACL. If that fails, the HAB sends back an error to the Client and drops the packet.

If the Client is authorized, the HAB composes a Consumer Stream Connection Accepted message like this:

```
Message-Type: Consumer Stream Connection Accepted
Delivery-Vector: <list of unicast endpoints for delivering data>
Symmetric-Key: <symmetric key>
```

The Client chooses an endpoint, connects to it, and starts sending data for the Consumer Stream, encrypted with the specified key.

4.6 Unicast transport protocols for Bionet

4.6.1 TCP

This one's easy.

4.6.2 AMS

NOTE: This section is incomplete.

Bionet can transit its traffic over the Asynchronous Message Service (AMS).

Each Bionet peer (that is, each HAB and Client) is an AMS Node.

Each HAB registers as the sole AMS Node in its own dedicated Unit of the message space, so that it can be explicitly addressed by applications; Clients all register in a common "Clients" Unit.

When HABs start up and shut down - and when Clients start up and shut down - all of the AMS nodes are informed of these events. But there aren't large numbers of HAB or Clients, and they don't start up or shut down frequently, so this configuration traffic is insignificant. The "application code" of each AMS node (a HAB implementation or a Bionet Client implementation) is notified of these events. An application can use a private mapping of HAB IDs to AMS Unit number in order to remember the specific AMS Node IDs of the HABs it is interested in.

Bionet Client nodes "invite" messages on two AMS subjects: NodeList and ResourceUpdate. HABs "invite" messages on three AMS subjects: NodeListRequest, ResourceUpdateRequest, SetResource.

Bionet Client nodes send messages privately (they do not publish) on subjects NodeListRequest, ResourceUpdateRequest, and SetResource; they send these messages to specific HABs, by AMS node number. The content of a ResourceUpdateRequest names the Node/Resource in which the application is interested. HABs receive these messages and use them to manage their own private, Bionet-level subscription lists or -- in the case of SetResource -- to effect changes in their sensor networks.

When Bionet Node lists change and Resource values change, the HABs send NodeList and ResourceUpdate messages privately to the application nodes that have asked for the newly available application.

When an application terminates, all HABs from which it has been receiving information are automatically notified and can remove the affected Bionet-level subscriptions from their private subscription lists.

Each Unit has its own Registrar, so there's one Registrar for all the Clients, and one for each HAB. Each HAB's Registrar would likely reside on the same computer as the HAB itself, so heartbeat traffic between them would never hit the net. They all share a single Configuration Server.

5 Bionet over Multicast

Bionet can transit its traffic over IP Multicast. Multicast traffic is restricted to unreliable datagrams. This can be very efficient when the underlying network has high reliability, but requires application-level efforts to improve end-to-end reliability when the underlying network is unreliable. The Bionet protocol has been designed to make the communications more reliable using retransmissions, heartbeats, etc.

5.1 Node List

Each HAB maintains a Node List inventorying all the Nodes it currently is presenting to the Bionet network. The Clients that subscribe to the HAB's Node List receive copies of this list. Bionet tries to keep all subscribers' Node List copies as correct and up to date as possible.

This is accomplished by a pub/sub mechanism to inform Clients of changes as they happen, a periodic “heartbeat” to inform Clients of the current Node List state, and a request/reply mechanism to synchronize Clients' Node Lists if they miss published updates.

5.1.1 State diagrams

5.1.1.1 Node List Subscription: Client state diagram

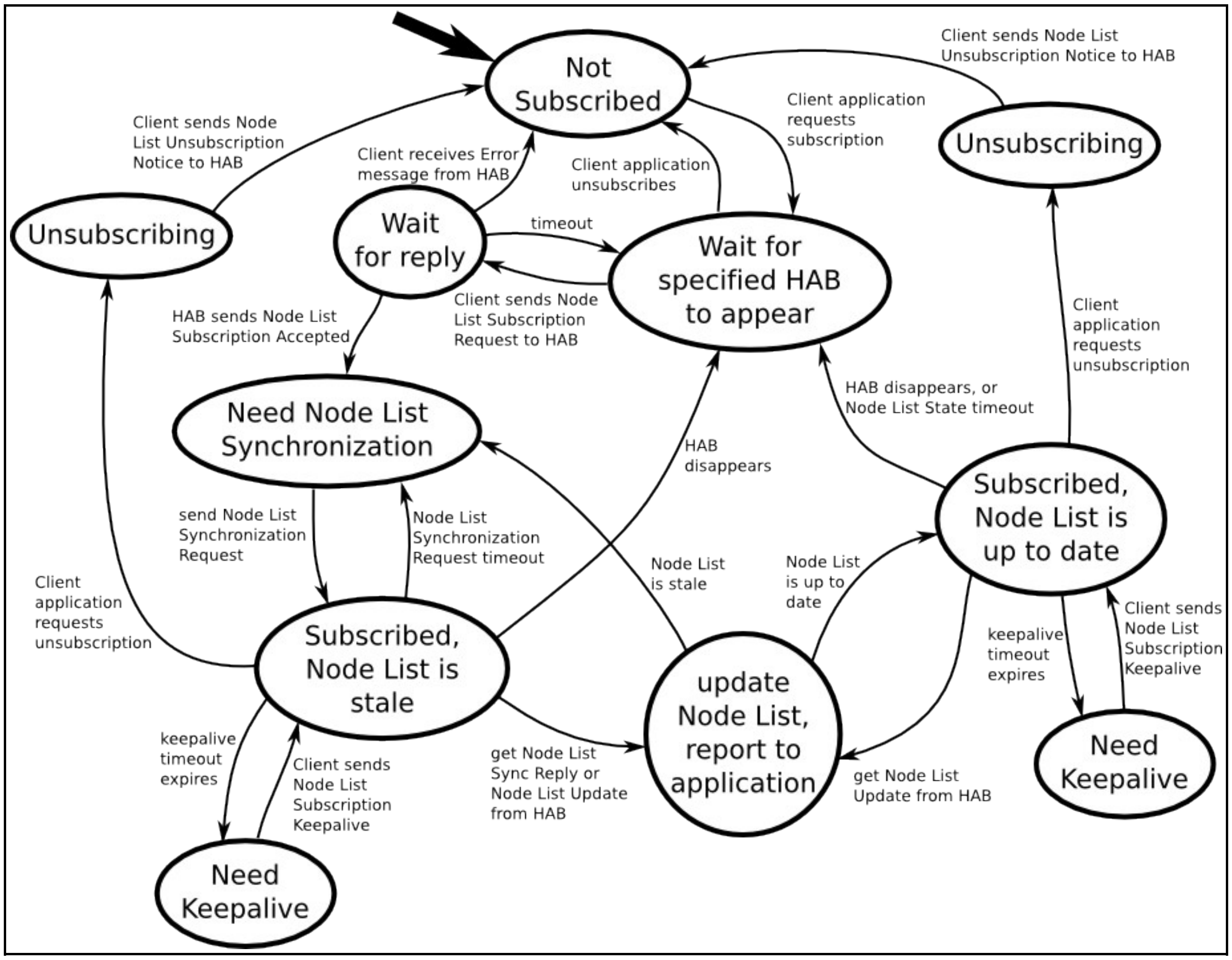
Not Subscribed
Client sends Node List Unsubscription Notice to HAB
Unsubscribing
Client application requests subscription
Client sends Node List Unsubscription Notice to HAB
Client receives Error message from HAB
Client application unsubscribes
Unsubscribing
Wait for reply
timeout
Wait for specified HAB to appear
Client sends Node List Subscription Request to HAB
HAB sends Node List Subscription Accepted
Client application requests unsubscription
HAB disappears, or Node List State timeout
Need Node List Synchronization
HAB disappears
Subscribed, Node List is up to date
send Node List Synchronization Request
Node List Synchronization Request timeout
Node List is stale
Node List is up to date
Client application requests unsubscription
Subscribed, Node List is stale
update Node List, report to application
keepalive timeout expires
Client sends Node List Subscription Keepalive
keepalive timeout expires
Client sends Node List Subscription Keepalive
get Node List Sync Reply or Node List Update from HAB
get Node List Update from HAB
Need Keepalive
Need Keepalive

Illustration 4: Node List Subscription: Client state diagram

5.1.1.2 Node List Subscription: HAB state diagram

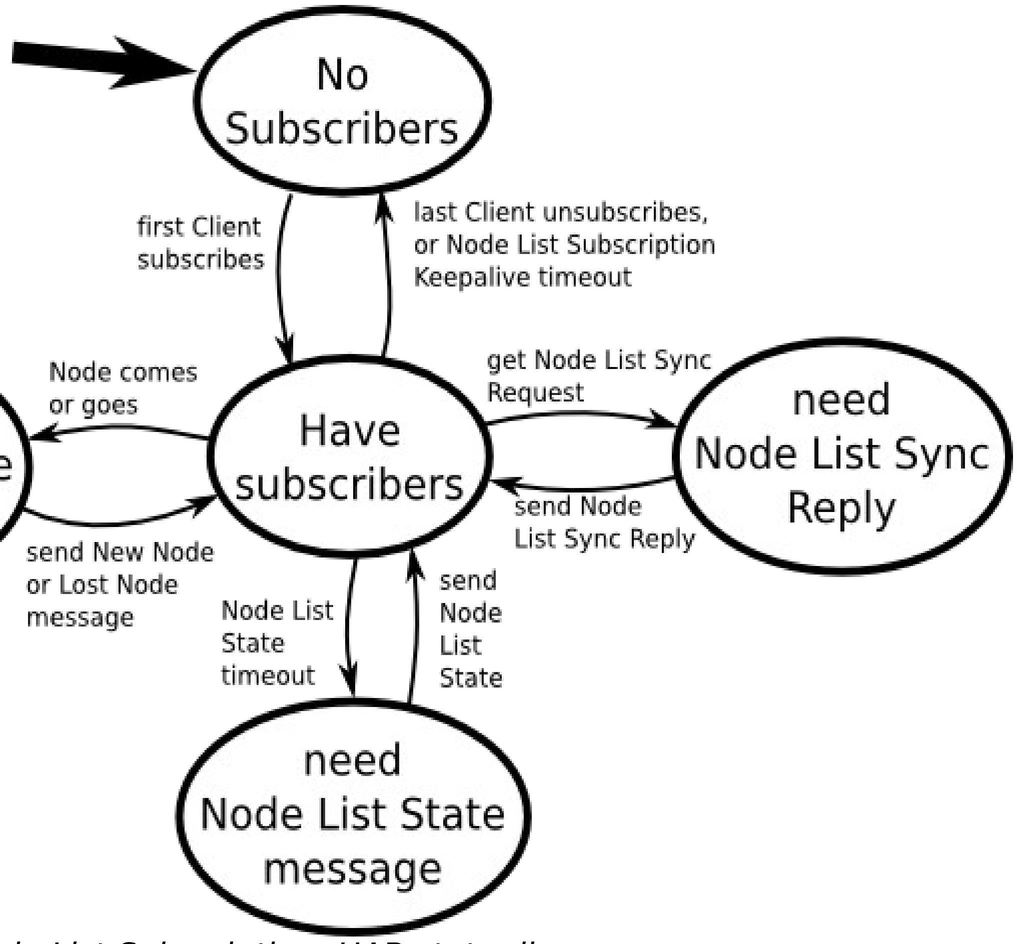
No Subscribers
first Client subscribes
last Client unsubscribes, or Node List Subscription Keepalive timeout
need Node Update message
Node comes or goes
Have subscribers
get Node List Sync Request
need Node List Sync Reply
send Node List Sync Reply
send New Node or Lost Node message
Node List State timeout
send Node List State
need Node List State message

Illustration 5: Node List Subscription: HAB state diagram

5.1.2 Registering a Node List Subscription

When a Client wants to subscribe to a HAB's Node List, it composes a Node List Subscription Request message like this:

```
Message-Type: Node List Subscription Request
To: HAB
Client-public-key: <Client's public key>
```

The message is signed with the Client's private key and unicast to the HAB. The HAB verifies the signature and checks the public key against the "Node List: HAB-Type.HAB-ID" ACL. If any of that fails, it unicasts back an error to the Client and drops the packet.

If the checks pass, the HAB composes a Node List Subscription Accepted message like this:

```
Message-Type: Node List Subscription Accepted
To: Client
Multicast-Address: IP address and UDP port number
```

```
    Symmetric-Key: <symmetric key>
```

This packet is signed with the HAB's private key, encrypted with the Client's public key, and unicast back to the Client. The Client initializes its Node List copy to be empty, joins the specified multicast channel, and begins a Node List Synchronization conversation (described below).

5.1.3 Publishing a change to the Node List

When the HAB notices that a Node has come or gone, it publishes a New Node message or a Lost Node message, respectively.

5.1.3.1 New Node

The HAB adds the new Node to its Node List and updates the Node List Fingerprint.

It then composes a New Node message like this:

```
    Message-Type: New Node
    To: Node List Multicast Group
    From: HAB
    Timestamp: <timestamp>
    Node: <complete description of the new Node>
    Node-List-Fingerprint: <HAB's Node List Fingerprint>
```

The HAB signs the message with its private key, encrypts it with its Node List Subscription's symmetric key, and publishes it to its Node List multicast address.

When a Client receives a New Node message it validates the HABs signature and decrypts the message.

If the Node already appears in the Client's Node List it reports the old existing Node as lost and removes it from the list.

The Client adds the new Node to its Node List and reports it to the application. It updates its Node List Fingerprint.

If the Client's Node List Fingerprint matches the HAB's advertised Node List Fingerprint, then the Client's Node List is up to date and the transaction is complete.

If the Client's Node List Fingerprint does not match the HAB's advertised Node List Fingerprint, then the Client's Node List is out of date, and the Client initiates a Node List Synchronization conversation (described below).

5.1.3.2 Lost Node

The HAB removes the lost Node from its Node List and updates the Node List Fingerprint.

It then composes a Lost Node message. The Lost Node message has this content:

```
    Message-Type: Lost Node
    To: Node List Multicast Group
    From: HAB
    Timestamp: <timestamp>
    Lost-Node-Fingerprint: <Node Fingerprint of the lost Node>
    Node-List-Fingerprint: <HAB's Node List Fingerprint>
```

The HAB signs the message with its private key, encrypts it with its Node List Subscription's symmetric key, and publishes it to its Node List multicast address.

When a Client receives a Lost Node message it validates the HABs signature and decrypts the message. If the Node appears in the Client's Node List it reports the Node as lost, removes it from the list, and updates the Node List Fingerprint.

If the Client's Node List Fingerprint matches the HAB's advertised Node List Fingerprint, then the Client's Node List is up to date, and the transaction is complete.

If the Client's Node List Fingerprint does not match the HAB's advertised Node List Fingerprint, then the Client's Node List is out of date, and the Client initiates a Node List Synchronization conversation (described below).

5.1.4 Node List State

The HAB sends out Node List State packets as a kind of heartbeat to continuously inform Clients of the true Node List. The HAB sends Node List State packets frequently when the Node List has just changed, and infrequently when the Node List has remained unchanged for a long time, using exponential backoff with a minimum period of a second or so and a maximum period of maybe a minute or so.

The Node List State packet looks like this:

```
Message-Type: Node List State
To: Node List Multicast Group
From: HAB
Node-List-Fingerprint: <HAB's Node List Fingerprint>
```

The message is signed with the HAB's private key and encrypted with the Node List's symmetric key.

When a Client receives a Node List State packet, it decrypts it and verifies the signature. If the HAB's Node List Fingerprint matches the Client's, no action is required. If the Node List Fingerprints differ, the Client initiates a Node List Synchronization conversation (described below).

5.1.5 Node List Synchronization

Node List Synchronization is an iterated process initiated by a Client, which hopefully concludes with all Clients' Node Lists totally up to date.

5.1.5.1 Node List Synchronization Request

When a Client believes that its Node List is stale, it composes a Node List Synchronization Request:

```
Message-Type: Node List Synchronization Request
To: HAB
Node-List-Fingerprint: <Client's Node List Fingerprint>
```

The Client encrypts the message with the Node List Subscription's symmetric key, and unicasts it to the HAB.

5.1.5.2 Node List Synchronization Reply

When the HAB receives a Node List Synchronization Request, it waits a short while for other Node List Synchronization Requests to come in.

The HAB compares the Clients' Node List Fingerprints with its own and composes a Node List Synchronization Reply message describing all Nodes the Clients need to drop and add in order to be up to date:

```
Message-Type: Node List Synchronization Reply
To: Node List Multicast Group
From: HAB
Lost-Nodes: <list of Node Fingerprints>
New-Nodes: <list of Nodes>
Node-List-Fingerprint: HAB's Node List Fingerprint
```

This message is signed with the HAB's private key, encrypted with the Node List's symmetric key, and sent to the Node List multicast group.

Each Client that receives the message decrypts it and verifies the HAB's signature.

If the Node List Fingerprint in the message matches the Client's Node List Fingerprint, no action is required.

If the Node List Fingerprint in the message differs from the Client's, the Client needs to update its Node List. The Client drops any Nodes in its Node List which match a Node Fingerprint in the Lost-Nodes field of the message, and adds any Nodes from the New-Nodes field which it does not already have in its Node List. Any Nodes in the New-Nodes field which already appear in the Client's Node List are ignored.

The Client updates its Node List Fingerprint. If the new Node List Fingerprint matches the one from the Node List Synchronization Reply message, the transaction is complete. If the Node List Fingerprints do not match, the Client sends another Node List Synchronization Request.

5.1.6 Node List Keepalive

All Clients that are subscribed to a HAB's Node List maintain independent Node List Keepalive timers. The timeouts are random in an interval of maybe 1 to 2 minutes. If a Client's Node List Keepalive timer expires, it sends a Node List Keepalive message to the multicast group:

```
Message-Type: Node List Keepalive
```

If a Client sees a Node List Keepalive from another Client, it resets its timer to a new random value.

If a HAB does not see any Node List Keepalive messages for some small multiple of the maximum Node List Keepalive timeout, it cancels all Node List subscriptions and stops sending out Node List updates.

5.1.7 Node List Unsubscription

When a Client loses interest in a HAB's Node List, it sends a Node List Unsubscription message.

```
Message-Type: Node List Unsubscription
Client-Key: <Client's public key>
```

The message is signed with the Client's private key and unicast to the HAB. The HAB removes the Client from it's list of Node List subscribers. If the list is now empty, the HAB stops sending out Node List Updates to the multicast group.

5.2 Resource Subscription

Resources have metadata (which is static) and data (which is dynamic). The Client learns the static metadata of a Resource when it learns about the Node that the Resource is a part of (via a Node List Subscription). The Client may then subscribe to the Resource, to be continuously informed of the Resource Value as it changes. Bionet tries to keep all Clients up-to-date with the latest Resource values of all the Resources they have subscribed to.

5.2.1 State diagrams

5.2.1.1 Resource Subscription: Client state diagram

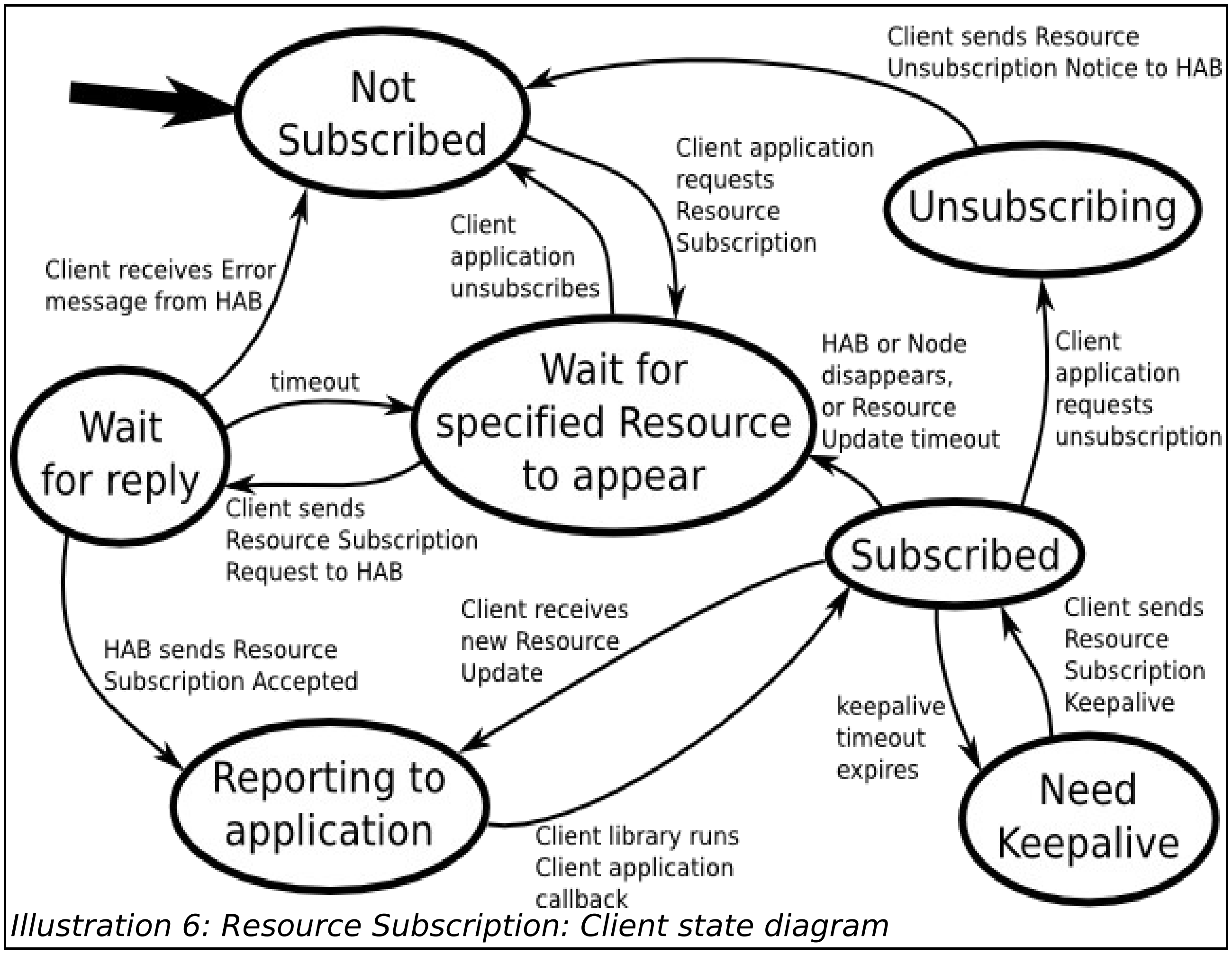
Not Subscribed
Client sends Resource Unsubscription Notice to HAB
Client application requests Resource Subscription
Unsubscribing
Client application unsubscribes
Client receives Error message from HAB
timeout
Wait for reply
Wait for specified Resource to appear
HAB or Node disappears, or Resource Update timeout
Client application requests unsubscription
Client sends Resource Subscription Request to HAB
Subscribed
Client receives new Resource Update
HAB sends Resource Subscription Accepted
Client sends Resource Subscription Keepalive
keepalive timeout expires
Reporting to application
Need Keepalive
Client library runs Client application callback

Illustration 6: Resource Subscription: Client state diagram

When a Client application wants to subscribe to a Resource, it makes a function call into the

Client library. The library registers the subscription request and returns to the application. At this point the application is free to do other things, and the library operates asynchronously to perform the subscription related tasks.

The library waits for the specified Resource to become available (this may involve waiting for the HAB to join the Bionet network, then subscribing to its Node List, then waiting for the Node to join).

When the Resource is available, the library sends a Resource Subscription Request to the HAB. The HAB sends a Resource Subscription Accepted message to the Client, and the subscription is established.

When a HAB sends Resource Update messages whenever a new value is available; if no new value is available the HAB periodically retransmits the most recent one. Each Resource Update message has a Sequence Number; retransmissions of old Resource Updates use the original Sequence Number, new Resource Updates increment the Sequence Number. The Client library only calls the applications Resource Update callback for Resource Updates with new Sequence Numbers. The Sequence Number is also used to notice when the Client has missed Resource Updates.

All Clients that are subscribed to a particular Resource conspire to provide a periodic Resource Subscription Keepalive message to the HAB, so the HAB can detect if all subscribers have crashed. Each Client has a random timeout within a predetermined range, between one and two minutes perhaps. Whenever a Client sees a Resource Subscription Keepalive from another Client, it resets its timeout to a new random value in that range. When the timeout expires, the Client sends a Resource Subscription Keepalive itself. If the HAB doesn't get a Resource Subscription Keepalive within a small multiple of the upper end of the random timeout range, it assumes all Clients have crashed and it stops publishing Resource Updates.

When a Client unsubscribes in an orderly fashion (as opposed to crashing), it sends a Resource Unsubscription Notification to the HAB.

5.2.1.2 Resource Subscription: HAB state diagram

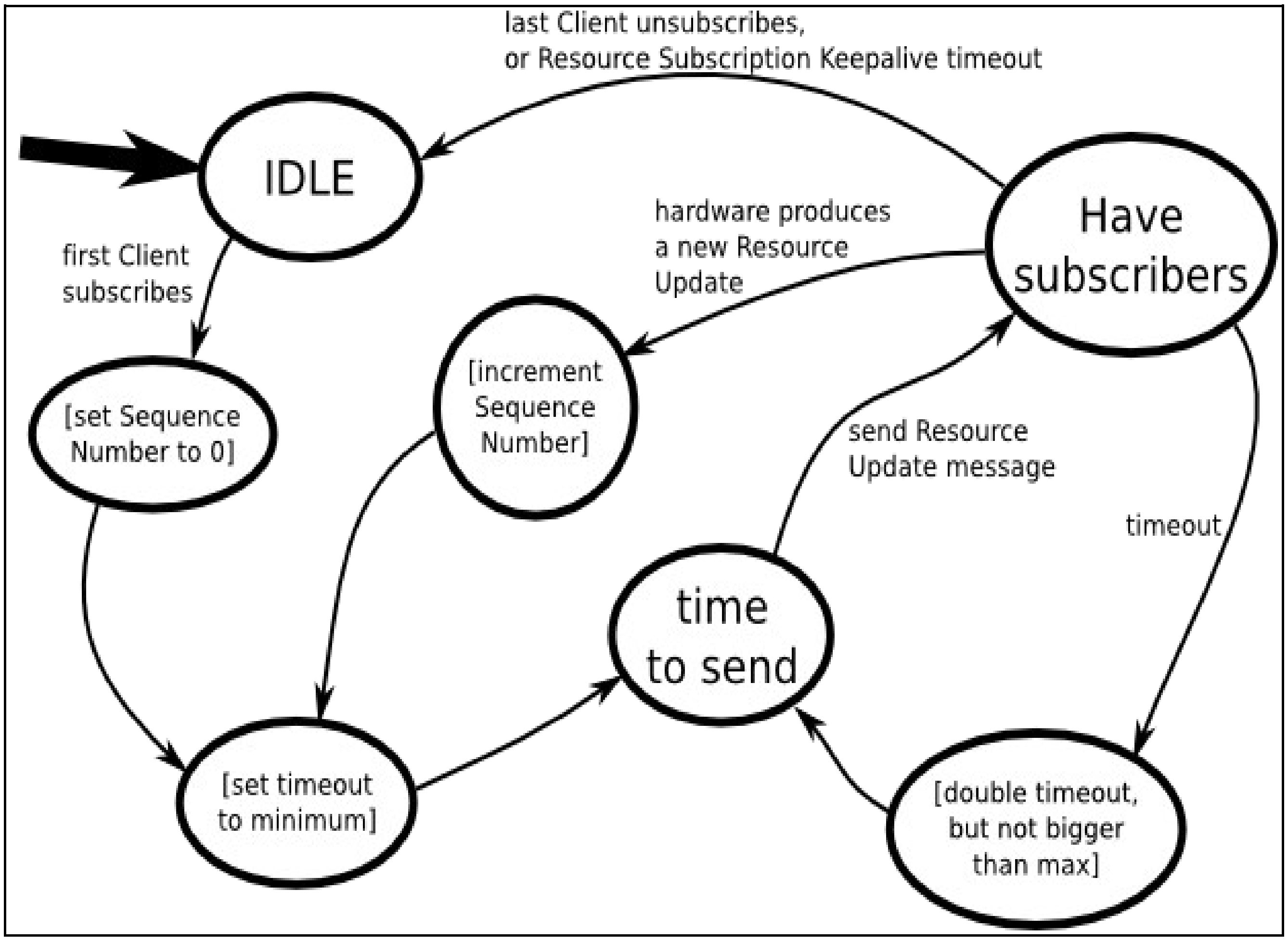
last Client unsubscribes,
or Resource Subscription Keepalive timeout
IDLE
first Client subscribes
[set Sequence Number to 0]
hardware produces a new Resource Update
Have subscribers
[increment Sequence Number]
send Resource Update message
timeout
time to send
[set timeout to minimum]
[double timeout, but not bigger than max]

Illustration 7: Resource Subscription: HAB state diagram

When a HAB has registered subscribers to a particular Resource, it publishes Resource Updates whenever the value of the Resource changes, and periodically when there are no changes. The periodic retransmissions of the most recent Resource Update act both to inform Clients that may have missed the original, and also as a kind of heartbeat letting Clients know if the HAB has crashed. The time between Resource Update retransmissions starts out very low when the value has just changed, and increases exponentially to some maximum.

5.2.2 Registering a Resource Subscription

When a Client wants to subscribe to a Resource, it composes a message like this:

```
Message-Type: Resource Subscription
To: HAB
Resource: <Node-ID.Resource-ID>
Client-Public-Key: <Client's public key>
```

The message is signed with the Client's private key and unicast to the HAB.

The HAB verifies the signature and checks the public key against the Resource's read-permission ACL. If any of that fails, it unicasts back an error to the Client and drops the packet.

If the checks pass, the HAB composes a Resource Subscription Accepted message like this:

```
Message-Type: Resource Subscription Accepted
To: Client
Multicast-Address: IP address and UDP port number
Symmetric-Key: <symmetric key>
Resource-Name: HAB-Type.HAB-ID.Node-ID.Resource-ID
Sequence-Number: <current Resource Sequence Number>
Value: <current Resource Value>
Timestamp: <current Resource Timestamp>
```

Note that this message includes the current value of the Resource. The message is signed with the HAB's private key, encrypted with the Client's public key, and unicast back to the Client.

The Client reports the initial Resource update to the application, joins the specified multicast channel, sets a Resource Subscription Keepalive timer, sets a Resource Update heartbeat timer, and listens for traffic.

5.2.3 Resource Updates

Resource Updates serve two purposes. They inform subscribers of the most recent Resource value, and they inform subscribers that the HAB is still running.

5.2.3.1 The initial Resource Update transmission

When a HAB produces a new Resource Value (and Timestamp), it sends a Resource Update message:

```
Message-Type: Resource Update
To: Resource Multicast Group
Resource-Name: HAB-Type.HAB-ID.Node-ID.Resource-ID
Sequence-Number: <Sequence Number>
Value: <Resource Value>
Timestamp: <Resource Timestamp>
```

The message is signed with the HAB's private key, encrypted with the Resource's symmetric key, and published to the Resource's multicast group.

When a Client receives a Resource Update message, it decrypts it and validates the signature.

The Client resets the Resource Update heartbeat timer.

If it has seen this Sequence Number before, it silently drops the packet.

If the Sequence Number is larger than the previously seen Sequence Number, the Resource update is reported to the application, and the transaction is complete.

5.2.3.2 Resource Update retransmissions

When a HAB has no new Resource updates to publish, it continuously re-publishes the most recent Resource update, with exponential backoff. This increases the likelyhood that all Clients eventually get informed of Resource Updates.

When a new update to a Resouce becomes available, the previous update is discarded, and the period between retransmissions is reset to its minimum.

Exact minimum and maximum retransmission periods are TBD, but the minimum might be something like 1 per second, and the maximum 1 per minute.

5.2.4 Resource Subscription Keepalives

All Clients subscribing to a particular Resource conspire to sends periodic Resource Subscription Keepalive messages to the HAB. If the HAB misses too many consecutive Resource Subscription Keepalives, it assumes that all Clients have crashed or been disconnected, and it ceases publishing Resource Updates.

Each Client generates a random timeout within some range (perhaps between one and two minutes). Whenever a Client sees a Resource Subscription Keepalive, it cancels its running countdown and starts a new countdown with a new random timeout. When the timeout expires, the Client sends its own Resource Subscription Keepalive on the multicast channel.

The Resource Subscription Keepalive message looks like this:

```
Message-Type: Resource Subscription Keepalive
To: Resource Multicast Group
Resource: <Resource Name>
```

5.2.5 Resource Unsubscription

5.3 Resource Setting

Clients can request that a HAB set the value of a Resource. This only works on Resources whose Access Mode is “Read/Write”. Each Resource has its own Write ACL.

5.3.1 Resource Setting Request

To command a Resource to a new value, a Client composes a Resource Setting Request message:

```
Message-Type: Resource Setting Request
To: HAB
From: <Client's public key>
Resource: Node-ID.Resource-ID
New-Value: <new Value>
```

The message is signed with the Client's private key, encrypted with the HAB's public key, and unicast to the HAB.

When the HAB receives a Resource Setting Request, it decrypts it, verifies the Client's signature, and verifies that the Client's public key is on the Resource's Write ACL. If any of this fails, it unicasts an error message to the Client and drops the packet.

If the verifications pass, the HAB relays the request to the specified Node. It attempts to verify that the Node accepted the request and implemented the command (if the sensor network and

Node support such verifications).

If the HAB reasonably believes that the command has been implemented, it transmits a Resource Update to the multicast group using the normal mechanism described above.

5.4 Streams

Streams behave differently depending on if their Direction is Producer or Consumer.

5.4.1 Producer Streams

Clients can connect to Producer Streams much like they subscribe to Resources.

When a Client wants to connect to a Producer Stream, it composes a message like this:

```
Message-Type: Stream Connection Request
Node-ID: <Node-ID>
Stream-ID: <Stream-ID>
Client-Public-Key: <Client's public key>
```

The message is signed with the Client's private key, encrypted with the HAB's public key, and unicast to the HAB.

The HAB verifies the signature and checks the Client's public key against the Stream's read-permission ACL. If any of that fails, it unicasts back an error to the Client and drops the packet.

If the checks pass, the HAB composes a Producer Stream Connection Accepted message like this:

```
Message-Type: Producer Stream Connection Accepted
Multicast-Address: IP address and UDP port number
Symmetric-Key: <symmetric key>
```

The message is signed with the HAB's private key, encrypted with the Client's public key, and unicast back to the Client.

The Client joins the specified multicast group, sets a Stream Connection Keepalive timer, sets a Stream Update heartbeat timer, and listens for traffic.

5.4.2 Consumer Streams

Clients can provide data for Consumer Streams. This is somewhat like providing Resource Settings, except continuously and generally at higher bandwidth.

When a Client wants to connect to a Consumer Stream, it composes a message like this:

```
Message-Type: Stream Connection Request
Node-ID: <Node-ID>
Stream-ID: <Stream-ID>
Client-Public-Key: <Client's public key>
```

The message is signed with the Client's private key, encrypted with the HAB's public key, and unicast to the HAB. Note that the Stream Connection Requests for Producer and Consumer Streams are identical.

The HAB verifies the signature and checks the Client's public key against the Stream's write-permission ACL. If any of that fails, it unicasts back an error to the Client and drops the packet.

If the checks pass, the HAB composes a Consumer Stream Connection Accepted message like this:

```
    Message-Type: Consumer Stream Connection Accepted
    Delivery-Vector: <list of unicast endpoints for delivering data>
    Symmetric-Key: <symmetric key>
```

The message is signed with the HAB's private key, encrypted with the Client's public key, and unicast back to the Client.

The Client chooses a unicast endpoint, connects to it, and starts sending data for the Consumer Stream, encrypted with the specified key.

6 Delay Tolerant Networking

Bionet as described up to this point operates over a fast IP network. The extension of Bionet to operate over delayed networks is a future work not addressed by this version of the design

In a “network of Bionetworks” model, each fast, low-latency local IP network hosts a Bionet network. Bionet network are uniquely identified by a Network ID. The Network ID is prefixed to all Bionet Names, to make them globally unique (not just unique within their fast local network).

Some thoughts on the general nature of Bionet-over-DTN follows.

6.1 Bionet Remote Gateways

This is an attempt to preserve the pub/sub nature of Bionet while extending it to work over delayed networks. This is accomplished by introducing a new type of peer called a Bionet Remote Gateway (BRG).

Should work over one-way links (no acks possible).

Prepend a Network-ID to all Names. "local" means "this one".

A "Bionet Remote Gateway" aka "Bionet DTN Gateway" has two interfaces: one on the local Bionet network and one on some long-haul delayed network. We model the DTN link as a point-to-point link between two BRG peers.

At startup, the Bionet Remote Gateway reads a list of persistent pre-positioned subscriptions out of nonvolatile storage. The BRG joins the Bionet network as a Client and registers the listed subscriptions with the appropriate HABs.

As the HABs publish data to the BRG, the BRG puts the Bionet data into BRG packets and sends the packets across the DTN to its peer or partner.

The BRG on the receiving side starts up and reads its own config file. The receiver's config file provides Node information about all the Nodes that it can receive Resource Updates for.

The receiver joins its local Bionet network as several HABs, proxying or masquerading as the HABs on the other end of the DTN link. It uses its own identity (key pair) and it presents its own identity certificate. The IC will have to be extended to include the Network ID. It joins the local Bionet network once for each HAB its masquerading for, giving unique unicast delivery vectors for each HAB (all with its IP address and different ports).

Clients can subscribe to available remote information by talking with the BRG. The set of available subscription topics is determined not by dynamic subscriptions registered on the receiving side, but by the pre-positioned, persistent subscriptions recorded on the sending side.

The DTN link is encrypted using pre-positioned symmetric keys. No key rotation is planned at this point.

6.1.1 Open issues

We need to specify the protocol between the BRGs.

We need some protocol to modify this subscription topic set.

6.2 BDM as the Bionet-over-DTN solution

An alternative to the BRG solution is to use the Bionet Data Manager (BDM) to bridge DTN links.

(For detailed information on the BDM, see the Bionet Data Manager Design document.)

In this model, each network that wants to “see” another Bionet network must provide a BDM process, and have the remote Bionet network's BDM process send its data to the local BDM process. Local Bionet processes that want to access data from the remote Bionet network then get it from the local BDM process.

The BDM (as currently envisioned) has four components:

- a storage back-end
- a Bionet Client personality that gets data from Bionet and stores it with the back end
- a BDM Sync personality that communicates with BDM peers and sends them data from the back-end store
- a BDM Server personality that proved local access to the data in the back-end store

To use BDM as the Bionet-over-DTN solution, we would add an optional fifth component to the BDM: a Bionet Proxy personality which would basically act like a BRG.