CHAPTER 6

Architecture for the Frontier

The general architecture of the Internet of Things (IoT) has been introduced in the preceding chapters, including concepts such as terse self-classified chirp protocols, structured tree networks, and the publish/subscribe framework. In addition, there has been an introduction to the key building blocks of the emerging Internet of Things: end devices, propagator nodes (and associated publishing agents), and integrator functions (and associated filter gateways). This chapter will explore the deeper architectural details of the Internet of Things, beginning with chirp formation at the edge of the network and continuing through to the propagation though the network, and finally to the implications of a publish/subscribe IoT world.

The key principle of the Internet of Things architecture is the segregation of networking cost and complexity to the propagator nodes, permitting much simpler components and architectures at the billions of simple end devices. These intermediate elements then bridge the gap between raw data chirps and big data meaning. With the assumption that the networking capabilities are in place within propagator nodes, individual end device appliances, sensors, and actuators may be implemented with a simple specialized language and vocabulary: the bare minimum necessary for what they were designed to do. Each type of device can use its own specialized format to chirp in its own dialect—no overarching standard common language is needed in every end device. Devices can remain simple, whereas propagator nodes (and publishing agents, if installed) can be arbitrarily complex.

A Necessary Alternative to IP

Beyond efficiency (large packet formats, etc.), there is a more fundamental reason to support a different transport protocol instead of couching a new description language inside the payload section of an IP packet. And the key factor is the need to support a one-to-many/many-to-one publish/subscribe framework.

Recall that the packet type ID in the IP packet header provides the information needed to drive traditional IP routing according to associated packet handlers. Adding a large number of new packet handlers, vocabulary, and protocols optimized to support the exploding variety of Internet of Things end devices to IP would pose challenges of scale, scope, and manageability. Routers would need software revisions to know how to route these new types of packets. That new software would in turn need to be deployed across the entire router network core and edge routers, including hundreds of thousands of legacy routers.

IP formats were originally designed for only the coarsest classifications of packet-type routing handlers; for example, voice, video, web browsing, and file transfer. Application-specific granularity (such as Devices ➤ Sensors ➤ Moisture ➤ Device-Type-A) cannot be easily expressed in a traditional format that was intended to address sender-oriented communications based on IP addresses and MAC IDs. If this type of data granularity were to be expressed within the payload section of an IP packet, the process of peering deep into each payload would slow down traffic unacceptably at each network device. These are the inherent limitations to IP sender-oriented, point-to-point traffic flow.

A Big Problem, and Getting Bigger

Although there are many expected classifications for appliance, sensor, and actuator types, this will be an evolving field into perpetuity. Providing specialized packet handlers within traditional IP routers to handle the routing needs of end device types yet undreamed-of is simply not practical. New types of end devices—and combinations of end devices, as illustrated in Figure 6-1— will constantly be added to the Internet of Things. There will also be the need for real-time localized control of semiautonomous relationships between sensors and actuators (see Figure 6-2), creating localized communities of machine-to-machine communications that are just beginning.

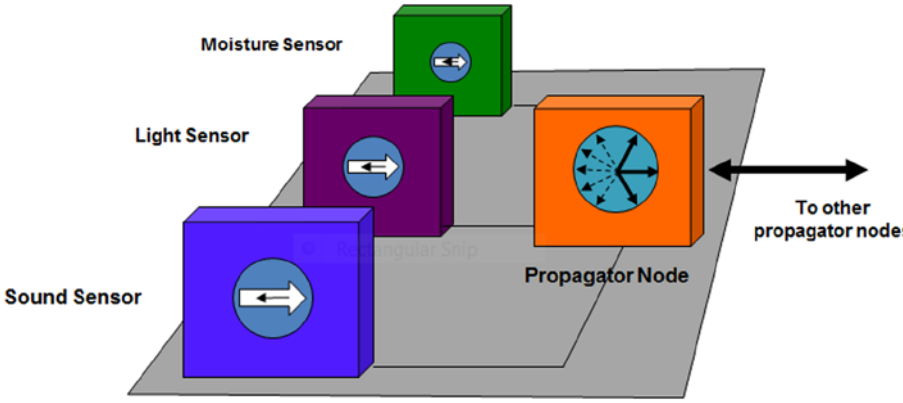

Figure 6-1. *Combinations of different sensor types within one physical package, each generating uniquely marked chirp packets, are just one example of the benefits of self-classified chirp protocols. In this example, an on-board propagator node could efficiently combine the chirp streams into "busloads" of small data for interpretation by one or more integrator functions*

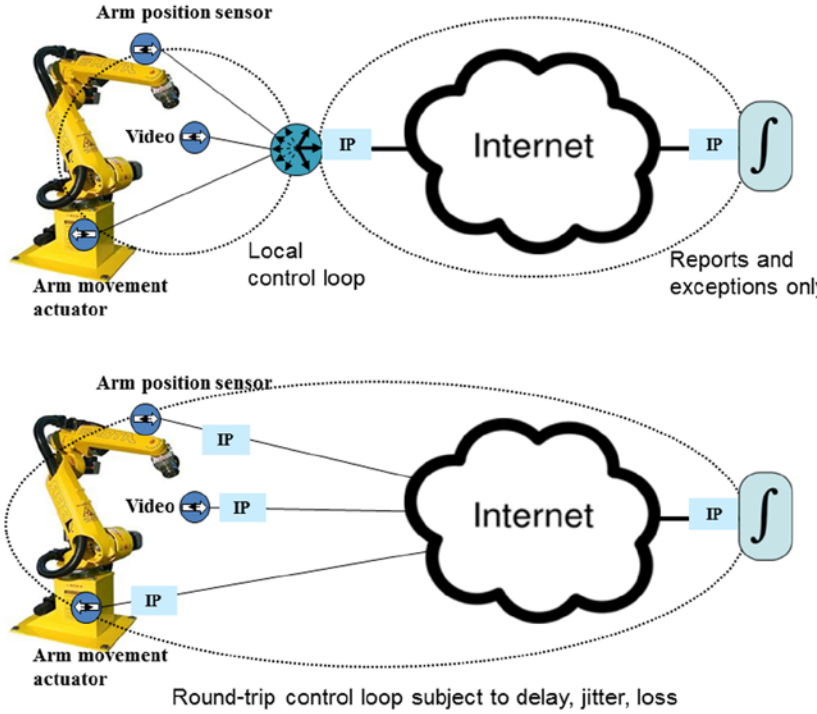

Figure 6-2. *In many emerging applications, local feedback from sensors and the corresponding commands to actuators will be needed in real time. A local control loop, illustrated here, uses data from positioning and video sensors to guide actuator movements (top). The risks, costs, and time associated with sending the real-time control traffic round-trip to a server(bottom) is not viable due to the risks of delay, jitter, and/or lost control packets (whether chirp-based or IP). But status reports and exceptions may still be reported to higher-level integrator functions while the local control loop manages real-time needs*

These new applications will thus require their own private small data streams and/or terse, tight local control loops. Standards-committee processes for IP and backbone routing take a long time and are understandably biased toward maintaining the status quo: that is, IP protocols used for nearly all communications (including the Internet of Things). Despite this, a more organic underlying architecture is needed that can be adapted rapidly to new end devices, independent of the techniques used for data transmission and analysis.

The primary reason for chirp–based end devices is their inherent simplicity and the fact that the chirp protocol may organically evolve to support device categories not yet dreamed of, let alone yet defined as to how they interact with humans and the world. Burdening these emerging publisher/subscriber relationships with the detritus and restrictions of a solely IP-based transport scheme is simply too small a canvas for the developers of these new products to create within (or to manage, build, and afford to produce once designed).

An Alternative Inspired by Nature

To solve problems of massive scale and generic broadcast infrastructures, nature uses a *receiver-oriented architecture*. Pollen "publishers" (plants) have no "receiver" (flower) address per se, nor do they know where their ultimate destination will be. A pollen category–based identification scheme is receiver-oriented: the self-categorized pollen simply travels in all possible directions; it is not destination-based or flowing in a predefined point-to-point relationship. It is the onus of the subscriber to accept (target flower) or reject (sneezing allergic human) the pollen. Publishing within classified categories in a natural environment (defined by genus and species) connects pollen to flowers in an inherently more efficient manner, but nature allows for evolution to take place over time. Thus, the "protocol" of pollen is externally marked with a self-classification recognized by receivers and may change over time.

A Protocol Based on Category Classifications

What would a similar extensible protocol look like within the chirp structures of the Internet of Things? In nature's DNA sequencing, there are strands of genetic code that are recognizable. Sometimes these specific genetic sequences serve as a marker that helps identify a distinct DNA sequence: relationships can be seen as the sequences repeat. Genetic fingerprinting is extensible as scientists learn more and more and can probe deeper into smaller sequences of information. The markers point to meaningful locations within the DNA sequence.

In nature's world of publish/subscribe, pollen is being published for subscriber flowers. Pollination is essentially a selective pattern match. The same logic will be applied to the IoT publish/subscribe world. In this case, rather than the wind distributing pollen promiscuously, a network of propagator nodes may use the structure of the chirp packet to direct data to an appropriate "receiver."

Chirp packets intentionally lack a target address; in a publish/subscribe world, the *receiver* chooses chirp streams and small data flows. So when these chirps are received by the first propagator node, what is needed to forward the arriving chirps in the appropriate direction? Recall that propagator nodes are aggregating and pruning chirps to form multichirp packets for transmission to the appropriate adjacent propagator node for eventual delivery to the integrator function(s).

Skeletal Architecture of Chirp Packets

A system that locates the end device publishers and integrator function subscribers efficiently and develops the correct routes is of common interest to both publishers and subscribers. Propagator nodes, as discussed previously, require some category description from the end devices to enable the matchmaking. What does this descriptor look like?

As an analogy, consider again bird chirps, the sounds of which may be organized based on the study of individual bird categories. Bird types may be identified by chirp/tune/melody. Hence those subscribers interested in melodies from doves can now receive those recordings, based on a bird category. The categories will have to support

different levels of granularity—some bird enthusiasts are interested only in doves near their homes. Hence the category field should be sufficiently flexible in design to support further drill-down.

In nature, melody/tunes and DNA structures incorporate marker strands of information that provide a common pattern across members of the category. The same is true for markers within Internet of Things chirp packets. These markers occur at specific locations and are of specific defined patterns.

A familiar example is the global telephone numbering system, in which information of increasing granularity is found in known standard locations. Country code, area code, exchange, and subscriber identification are progressively used to route the call to its final destination.

Within the Internet of Things, the final destination may not be known (again, due to the publish/subscribe nature of the IoT). So the chirps must be self-classified in a similar granular fashion to allow other network elements to act upon them.

For example, a given chirp category may have an 8-bit marker, which is always found in the fourth byte of the bit stream. (This pattern is indicated by the format of the chirp packet's offset marker, as shown in Figure 6-3).

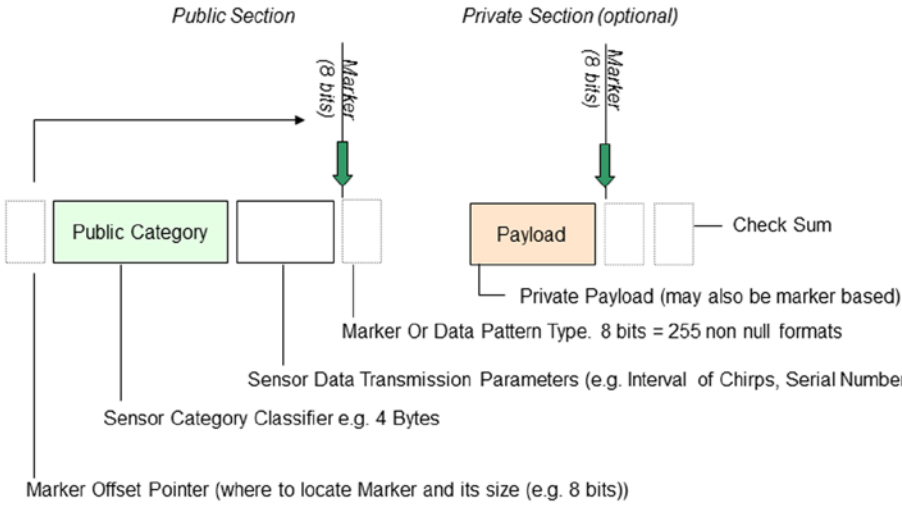

Figure 6-3. *The marker offset pointer allows IoT elements to recognize the locations of the chirp packet's public and private markers without examining more deeply within the packet, making routing and other networking decisions more rapid. The markers in turn provide information on how to interpret the end device's self-classified category and type*

One way of expressing this category classification is seen through an example. Consider a combination of a 4-byte classification and one additional marker byte of 8 bits. This can be expressed as 4.8(*XXXX*), where *XXXX* are more levels of granularity that may be gleaned from the 4 bytes by knowing the specific marker pattern format of the 8 bits and what that format entails. In this case, the 8-bit marker explains how to decode the 4-byte public classification. This will include the end device type (moisture sensor versus

street light, for example) and the way the 4 bytes of data are structured. The 4.8 pattern format alone would be sufficient information for a propagator node to make basic routing decisions (see below).

Additional information can be obtained from the value of the 8-bit marker. Consider an 8-bit marker pattern that is 1.1.1.1.1.1.1.1 (or 255). This value of 255 may indicate a format in which each of the preceding 4 bytes is a 1-byte classification subcategory. Thus, a 4-byte category may now be interpreted as A.B.C.D., where the letters occupy 1 byte each and indicate some subcategory. The complete interpretation of the category is thus 4.8.255.A.B.C.D.

The chirp packet will also contain the actual payload of the sensor values, but note that these have not been discussed so far. This is intentional, as it demonstrates that the propagator node may route quickly and efficiently on only the first bytes of data received without deeper examination of the chirp packet.

This enables a quick bit mask to look for all publishers in categories 4.8.255, and so on. Propagator nodes with internal publishing agents capable of acting upon further granularity in the chirp signature will need to access a reference that provides a map or implicit field markers for A.B.C.D within the category field. Thus, it can be imagined that all of the following provide successively deeper classifications of the chirp packet:

4.

4.8.

4.8.255

4.8.255.A

4.8.255.A.B

Thus, the propagator nodes, depending on their access to internal field data, can provide multiple levels of granularity in addressing (and may potentially act upon the complete chirp packet through a publishing agent that is aware of the meaning of the full address).

The simplest category of 4.8 may be sufficient for coarse aggregation: chirps of the same "feather" may be flocked together (see the "Scheduling the Bus" section that below). But additional levels of granularity in propagator node bus scheduling and routing are supported by considering more data.

Larger and infrequent buses might cover 4.8.XX categories, while smaller "shuttles" for more-frequently-requested data may specify precisely what is of interest; e.g., 4.8.255.A.B.C.XX. Chirp self-categorization thus drives the loading of multichirp–packet forwarding buses, their contents, and their frequency, at differing levels of granularity.

Note that A.B.C.D is distinct from B.A.C.D. In general, there are 4*4*4*4 or 255 non-null combinations for a four-letter vocabulary: A, B, C, D.

Obviously, the 255 combinations allowed provide tremendous flexibility in the way the 4-byte category is interpreted. Like DNA, the alphabet may be terse and small, but the patterns depicting the categories are not. An exceptional variety of content may thus be expressed within short chirp packets.

In fact, the category system is flexible enough that the simplest data payloads may be expressed within the public category *alone* with no separate payload. They would be very basic states expressible in a few bytes.

Individual Information within Chirp Signatures

Beyond category information, bird chirps also carry individual and private information. Nature's random number generator changes the individual birds chirp tonal qualities governing each bird. This serves as a form of identification. Thus, mother birds know each of their children's distinctive chirps, although all are using the same broad general chirp format and its associated shared vocabulary.

The Internet of Things counterpart of this sort of individual identification within the chirp packet is labeled "Sensor Data Transmission Parameters" in Figure 6-3. In combination with the "Sensory Category Classifier" seen in Figure 6-3, chirp identification parameters can include the following:

- Chirps with distinctively different patterns (i.e., tunes)
- Public category classification, including some specific (though not unique) end device identification; e.g., the last four digits of the manufacturer stock keeping unit (SKU) number of the device

Additional identification information is added by the connecting propagator node:

- **Lineage based:** For example, an end device associated with a kitchen propagator node
- **Location based:** For example, located in a kitchen, close to the toaster (derived from signal strength analysis, not a logical connection)

Note that the combination of a chirp tune (in this example, its last 4 SKU digits), its location, and its lineage *collectively* can define a distinct end device sensor or actuator. Although none is globally unique, the *combination* will likely be sufficiently distinctive for the vast majority of applications. The absolute uniqueness of the address *is not* required.

The combinations have inherent randomness because their constituent elements (e.g. transmission pattern of chirps) are random. They are not required to be unique, as are IP or MAC ID addresses, so there is no burden of maintaining a global database. Purely local "pretty good" distinction in the bird chirp is sufficient for the mother bird. By the same token, "pretty good" distinction for local end devices is sufficient for propagator nodes.

Note that individual data, while often in the private section, may also be present in the public section. Thus, some common types of end devices (e.g., temperature sensors) may not need a private section: the data may not need to be secured.

"Light" Error Detection and Security

The combination of marker and public category classification provides a first level of light error detection. For example, if the 8-bit marker described previously calls for a 4-byte category classification, but some other value is found instead, an error is recognized, and the chirp is discarded. Similarly, if the marker is corrupted and does not match the (correct) category classification, the chirp is likewise discarded. This is the reason why the marker occurs after the category classification within the chirp packet; it acts as a simple error-detection mechanism without creating any additional overhead.

Errors that occur elsewhere deeper in the chirp packet may elude this first level of error-checking, but any mismatches of markers and classification will eventually be detected. The presence of any propagator node that compares the sequences within the chirp stream will eventually result in this chirp being discarded. Because chirps are typically repetitive, the loss of this single corrupted chirp is not critical. Note, however, that corrupted chirps are being progressively pruned; often *before* the chirps are combined into IP packets.

Unlike the capabilities of IP packet headers, this light error detection allows a small number of errors to be propagated through part of the local network. But the savings in overhead for each chirp packet is well worth the small cost of handling some bad packets through portions of the network.

Generic Chirp Handling

The deeper chirp packet examination described above pertains primarily to propagator node networks containing publish/subscribe agents. If the propagator node has no publishing agent installed, small data flows are managed by the network topology and the arrow of transmission incorporated in the public marker: either toward integrator functions or toward end devices.

Here, the network topology of uplinks and downlinks (refer to Chapter 4) is being used to help move data toward an appropriate destination. Note that the directions can encompass both the propagator node topology and the parent IP–based network tree in a hybrid mesh network that incorporates both.

Incognito Chirp Transport

Some classification categories of chirps might have to travel incognito. That is, they expect propagators to rebroadcast them, potentially in all directions, until an appropriate publishing agent or integrator function discovers them.

"Incognito" chirp streams create the equivalent of a Virtual Private Network (VPN) within the IoT. They constitute a category that is indecipherable by nonproprietary publishing agents and integrator functions. Although they may be transported generically by the propagator node network, they typically could not pass through the chirp-to-IP interface as is. In the typical application, there will be a separate propagator node with a corresponding proprietary publishing agent somewhere on the "chirp" side of the network that has the "key" to interpret the private information within the chirp packets. From this, they may generate IP traffic that could traverse the global Internet to be acted upon by integrator functions also programmed to be part of the incognito network.

A "4.0" category chirp implies a marker at byte 4, but its length is not specified. Agents with bit mask filtering can locate such semi-incognito chirps because they know what the marker is. Note that the marker can be arbitrarily long or short. Short markers increase the occurrence of false positives with other marker types (e.g., the 4-bit marker 1.0.1.1 shares 4 bits with the 8-bit marker 1.0.1.1.0.0.1.1). Publishing agents that have this level of information can also glean other data from the packet melody/strands to filter out undesired or malformed chirp packets; these packets will not cross over to the IP network.

A “0.0” category chirp does not specify either the location or size of the marker. This is *completely* incognito, and the propagator node may continue to rebroadcast the chirp both up and down the propagator node tree until it reaches end devices, a publishing agent within a propagator node, or integrator functions of the network (depending on the arrow of transmission). Recall that native chirp devices have no access to the IP network except through propagator nodes, so IP traffic congestion is limited.

In some situations, a 0.0 chirp might want to specify the arrow of transmission and nothing else (e.g., up or down the tree). Because each category has its own vocabulary and language, privately defined 0.0 chirp families may choose to use a unique location in the chirp packet for the arrow of transmission. Languages defining the meaning of the data comprised of bit streams are both versatile and secure because they are generally receiver-oriented and do not require a deeper understanding within the propagator nodes.

IP–based end devices may also use category patterns as part of their data classification schemes. In that case, IP-based packet headers will specify the end device MAC ID or serial number within the payload along with the category classification. IP-based agents in the integrator functions or local to the IP interface of a publishing agent–equipped propagator node could then act on end device identification and category classification. Thus, a single integrator function may incorporate chirp streams aggregated into small data flows transmitted over IP and the traditional Internet, as well as more sophisticated end devices sending and receiving in native IP.

By the same token, end devices may include a specific IP address where they want their chirps to be sent in their private payload or public category type. The chirp interface of a publishing agent–equipped propagator node receives this chirp, which may be pruned and repackaged as needed for IP transmission to the specified address.

Transmission Agility Information within the Chirp

If chirping end devices share the same wireless medium (such as in Wi-Fi), one part of the public category section will also contain chirp transmission characteristics. In other words, the basic chirp structure must support network agility, even if a large majority of the end devices cannot act upon it *themselves*. Smarter, more agile devices can become aware that simpler chirp devices will be active at the intervals specified.

Thus, data related to when and how often the end devices chirp and what pattern they use (as in melody/tunes or rhythm) is needed by both propagator nodes and smarter agile end devices to ensure that elements of the network can anticipate and hear chirps distinctly and without collocated interference on the same “channel” from other devices. (Note that in nature, bird chirps are often interleaved; birds are aware of each other and may actively avoid transmitting simultaneously.)

This data also gives propagator nodes the option to shift smarter, more agile chirp devices to other times. Or the propagator node, after review of local client device transmission patterns, can forward a request for a change to the dipswitch settings of a simple end device. An indication of whether specific end devices support such flexibility is again part of the pattern marker. Thus, after some tuning, there may be sufficient distinction in the melodies so that propagator nodes can easily recognize individual end device “offspring” by their pattern of transmission.

Extensible, Nonunique, Pattern-driven

A broader view of the chirp architecture emerges. It contains patterns, defining other patterns, each of which provides a more refined level of detail. Defined levels of access to that detail can reveal:

- What type (category)of chirp is being transported?
- How often is this data published?
- What is its publishing frequency pattern? (Perhaps it is dynamic or it may need observation over time, implying learning and discovery.)
- What are the distinguishing features of individual chirp devices, such as serial number, location, and lineage?
- What is the information on the transmission pattern that enables agile devices to share the same medium without interference?

Note that all of this information is easily discerned by rudimentary bit masks—that is, *if* a particular pattern is known. One example is a propagator node that is instructed to look at bit location 13 in all 4.8.11 packets. If that bit is set to "1", it indicates a universal flag for "unit malfunction, type 1". The propagator node is required to convert that information into an IP packet and forward it to the manufacturer specified in another segment of the chirp packet.

The public section defines the chirp category needed for bus scheduling and packaging of packets (as noted previously). Without this category information, the propagator node would not know which direction to send the packets, as in which bus route to employ going up or down trees, and where to clone more packets for multicast transmissions when multiple subscribers exist.

The second, often private, section is the message: what a particular end device is saying and some (typically proprietary) information about this end device. It uses the same concepts as the public section, but it has its own markers and definitions of what those patterns signify and hence the location the implicit private field markers. The 4.8.11.A.B.C.D category family may use a completely different scheme for the *private* section than the 4.8.11.A.B.D.C category family.

The public and private parts of the chirp packet are separated by a publically known *public section end marker* that can be of variable lengths. For example, it may be 4 bits or 8, depending when whether 15 or 255 different types of (public) chirp patterns are needed.

Some default public markers will be provided through consensus or standards bodies and working groups (see Chapter 8). They will be reserved for common use by end device manufacturers of a specific category (e.g., all moisture sensor manufacturers might use a category such as A.B.C.D.E.F, a 6-byte category address).

Category Byte Size

Many simple devices may require only 1 byte for a category (255 variants) and another 4-bit marker for the pattern type (see Figure 6-4). Thus, each of the 255 category numbers may be interpreted up to 15 distinct ways. This allows for close to 212 interpretations of a

1-byte category field. Similarly, a 6-byte category field would allow for (248 -1) variants, each supporting 255 patterns (8-bit marker). The "genetic code" describing an end device category may be expressed in multiple ways using this extensible pattern based–format, which would permit additional categories to be defined over time.

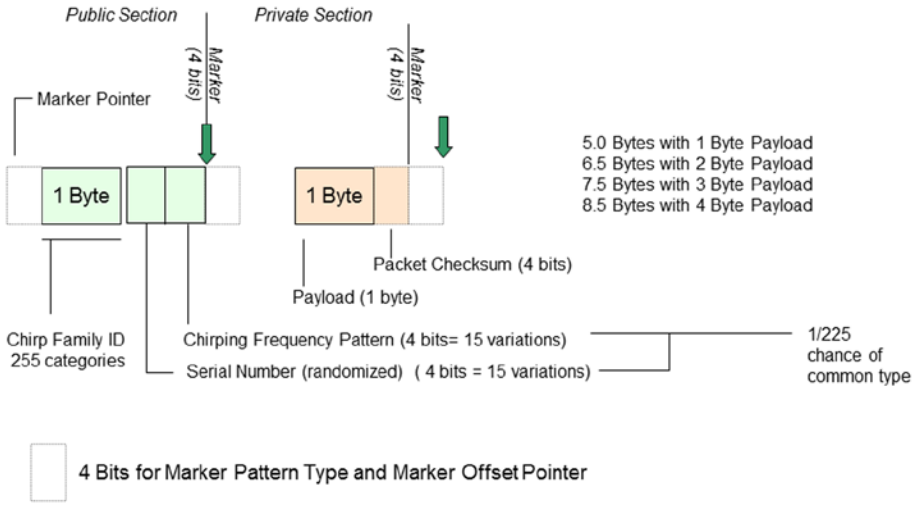

Figure 6-4. *Chirp packet size is variable; appliances, sensors, and actuators sending or receiving only small amounts of data may have a correspondingly low overhead applied to their chirp packets. Markers indicating chirp packet length are external to the data field to allow quick analysis by network elements and incorporate device type and classification*

For non-incognito (non-zero-byte) public sections, the marker type provides all the information needed to interpret it. The pattern defines where the content subsections/fields reside within the public section. Hence simple devices may use a larger public section to include data that is *also public*. Here, no private section is needed or used.

A zero-byte location is defined to mean that there is no public section. The marker type points to a data pattern that provides the information needed to interpret the private section, following the (empty) public section. The marker pattern is then used to interpret what follows generally as a payload. Thus, the flexible use of the marker pattern is supported beyond its initial intended use. A marker pattern and the associated classification of the data packet may together constitute an IP packet payload. This is relevant to IP-based sensor streams that prefer native IP connectivity over chirp-to-IP bridging through propagator nodes.

Marker Pattern Templates

Sharing the same marker type at the specified locations within the chirp packet engenders collaboration between manufacturers of the same sensor type. They may agree to jointly use a range of marker types (200–220, for example), which would be common

fields, but each may then use other fields (both in the public and private sections) to provide more detailed and/or secure information. A shared used marker pattern template emerges through this collaboration.

Creating a new marker type (say, 221) may not require the traditional central standards body review process because the repercussions are limited to that group of manufacturers. For example, introducing a new marker type in location 1, affects only the 1-byte public category users. Within that, it affects those who want to use the same marker pattern number. Contrast this with the challenges inherent in defining a new IP header format. IP headers must universally comply with IP requirements in order to be readable, with any change potentially affecting all users.

The marker template is therefore an organically evolving pattern-masking scheme that helps integrator functions delve deeper into the public section/category classification ID. As such, it loosely resembles IPv4 or IPv6, which are subsections of the entire IP address.

Note, however, that IP addressing is destination-based, so after the packet reaches its destination the payload is extracted. Then the information, perhaps still device-specific, must be device-abstracted. Next come pruning and aggregation in the generation of small data. The small data is now publishable within the distributed processing of big data servers (e.g. Apache Hadoop–based). It must now be inserted into the publish/subscribe framework of web-based services.

The situation is much simpler in the Internet of Things through the use of chirp category marker templates. Small data streams are generated closer to the end device source by propagator nodes, in which data can potentially have more real-time impact in tighter sensing-control-actuation loops. And because chirp-based traffic is category-based, finer granularity is simply a matter of loading the appropriate publishing agents at any level within the propagator node network or chirp-aware integrator functions that know how to look into IP encapsulations of aggregated small data.

The category section is a bit stream with contiguous fields, like strands in a DNA sequence. Knowing how to look into it helps better decipher the end device chirp category. But this requires more processing and is therefore intended for integrator function subscribers interested in finer granularity. As the chirp streams become aggregated into small data flows and move through the network, information continues to be disseminated with finer targeting to the interested integrator function subscribers, who can also drill down, if they prefer, by requesting broader category searches. As noted in Chapter 5, this can be through biasing of the publishing agents within propagator nodes that are so-equipped.

But even without a publishing agent within the propagator nodes, the lowest level of granularity needed is simply the marker location and its number. Hence, a byte 6, 4-bit marker, with value 1.0.1.1 is sufficient to have the chirp aggregated toward an appropriate agent for type 06.4.11. This may be a publishing agent in another propagator node or an integrator function.

Finer Control via Agents

At the first agent with an *understanding* of 6.4 buses, specialized 6.4.11 processes may peruse the category pattern to uncover two more subcategories, each of which might be specified by the pattern description to be 1 byte each. Such an expanded category

might be interpreted as 6.4.11.250.250. Subscribers willing to pay for this level of detail are alerted to the availability of a small data stream with that category classification. Thus end-device chirps can be very specific in terms of the type of agent they may be transmitted toward using a variable pattern template structure.

Publishing agents in the propagator node path allow chirp streams of specific types to somewhat manage the network that carries them because manufacturers can decide where those agents are placed along the route, starting with 6.4.11 and becoming progressively finer.

The "bus" transmission schedules of aggregated packets are now driven by the amount of traffic and any delivery timing specifics set by the subscribing integrator function(s). The size and content of the small data streams are being managed to ensure timely delivery in dynamically changing scenarios. This becomes a more tractable problem as more exploration into the chirp category is possible closer to the chirp publishers. However, having pattern matching agents 6.8.001 through 6.8.255 (8-bit marker) resident at a local propagator node requires more CPU processing, which may be suitable for an enterprise application propagator node but it is overkill for the home.

Hence, multiple types of propagator nodes emerge, some perhaps to generate small data streams for specific category types. Or SIM cards slots may be provided, so that additional categories of chirp packet–handling publishing agents may be supported. Some of these bus-handling specializations will be secured to specific hardware; others may be software agents/apps.

Scheduling the Bus

The "bus-loading" process is roughly akin to placing human passengers on the proper bus route to reach the appropriate destination. For schoolchildren on the first day of classes, the available information might be the passenger's name and grade from a nametag. While the child does not know the correct route or even the destination address, the teacher supervising bus loading has the requisite knowledge of the routing network to make correctly decide upon which bus to place the passenger.

At the propagator node, then, arriving chirp packets will be collected and then be directed to the "bus" (transmission path) best suited for them. This must be determined largely by public information provided by the chirp packet markers. (If there are publishing agents deployed within the propagator node network, chirp packets may be examined further to determine how they should be forwarded or discarded.)

Ultimately, only the subscribing receiver, typically an integrator function, will look at the *full* chirp to determine if there is information sought by that receiver. The propagator nodes need only basic information from the chirp packet markers to make initial routing decisions. Further, there can be small changes in the data; it need not be error-free. As long as the markers are not corrupted, the faulty data will still find its way along the network.

Propagators simply need to know what direction to send the data – up or down the network tree in light of the transmission "arrow" found within a marker. This is not complex in a tree structure with O(n) routing (see sidebar "Why Trees Scale" in Chapter 4). Recall, this is not a peer-to-peer network, requiring an O(n2) computation of the routing paths, as suggested by traditional sensor networks, e.g. ZigBee. Thus, the direction (up/down) suffices in tree structures. And the direction for an end device chirp packet should point to where subscribers are.

Routing on Category Classifications

The shared routing table within the meshed propagator node network keeps track of where the clients are, includes chirp devices and publishing agents. Some chirp routing agents may be on the chirp-to-IP bridge, and capable of securely accessing the entire category fingerprint, perusing the contents and decide what to do with it.

The efficient path of the chirp data is thus gated, filtered and then redirected at progressively finer sieves, akin to Zip code classifications for postal mail. Letters that fit "standard" patterns (size and weight) are processed efficiently. Others will be dealt with after the simple stuff is completed—this is how greedy algorithms work. The price paid for the flexible chirp format is that nonstandard package types will emerge and must be handled, albeit less efficiently.

For maximum efficiency in local pruning and aggregation, it is best to place publishing agents closer to the end devices' raw published data streams. Here the publishing agents have more control over what is forwarded and how. A subscription model would defray the cost of transport and pruning.

In addition to the task of aggregation in building small data stream buses, propagator nodes may also be required to perform pruning in response to their subscriber preferences. Traffic flowing upward from remote moisture sensors in the wine country in France to an Amazon–hosted cloud service in the United States could well be small, but given the number of such sensors, the IP traffic is significant. IP traffic is not free; some means to control what is sent over IP is needed—specifically, the pruning of repetitive data close to its source (as opposed to at the integrator function).

As an example, in one network there might be a handful of 4.8.XX chirp category end devices; others are all 2.4.XX or 6.8.XX. It would make sense to move the 4.8.XX agents to a propagator node that handles more 4.8.XX buses. A 4.8.XX bus central "hub" emerges, at least temporarily, based on the center of gravity where 4.8.XX end devices and their subscribers are located. Some chirps may have more hops to travel; but by economies of scale, 4.8.XX bus deliveries and scheduling become easier and less costly.

Dynamic loading on the network is examined by the propagator nodes forming the hybrid mesh tree (of both IP and chirp devices) from the IP connection downward toward the chirp end devices. System administrators are notified as to the best locations to locate publishing agents on the propagator nodes. This will alter the data paths and streamline flow. Further, if the publishing agent is mobile (as in not locked-in to a particular physical device), the network can automatically move the publishing agents to optimize overall traffic flows. This is akin to changing the physical network topology to meet changing latency and throughput requirements.

Managing the Load

Both the physical network topology and the logical network (based on where publishing agents and integrator functions reside) eventually stabilize and learn to adapt the topology to provide stable, tunable bus forwarding schedules and routes for the small data streams.

Chirps may be merged, pruned, or aggregated at each propagator node along the path, based both on network topology and (if present) publishing agent biasing by integrator functions. This is necessary for a variety of reasons: some repetitive data may be discarded, new paths discovered, rerouting around failures and congestion, termination of subscriptions by integrator functions, and so on. The "publish" and "subscribe" sides of the Internet of Things are thus in dynamic alignment.

Propagator Node Networks and Operation

The foregoing chirp architectures and routing algorithms are acted upon by an interconnected network of propagator nodes and the traditional Internet, as introduced in Chapter 4. For the reason outlined there, tree-like structures are chosen as the most scalable and efficiently self-organizing structure for these networking elements of the Internet of Things. The propagator node network connects the end devices at the frontier of the Internet of Things without requiring IP connectivity end to end.

Trees are older than man and have a highly evolved networking structure that is both efficient and adaptive. The structure is recursive: any part of the tree replicates the same structure. The underground roots are an inverted tree and branches are miniature horizontal trees, all connected through the trunk. In a network of trees, some are "rooted" to the tree trunk; others through relay nodes. The logical and physical network of branches all follow one simple rule: the "uplink" (the head of the branch) is always one. A pitch fork branch (one with three roots to the tree trunk) would be considered a freak of nature. It is this simple rule—one uplink only—that ensures O(n) routing. Scalable networks are thus possible.

A Tree Grows in the IoT

Nature's tree structure is emulated in the emerging IoT architecture. For example, consider the propagators P0, P1, P2, and P3, as shown in Figure 6-5. P0 is the "root" node because it has access to the IP network. P0-P1-P2 form a "string of pearls" relay for chirp clients C3 and C4. C3 and C4 both share the lineage P0.P1.P2 and hence are identified as siblings. This lineage becomes part of their identity.

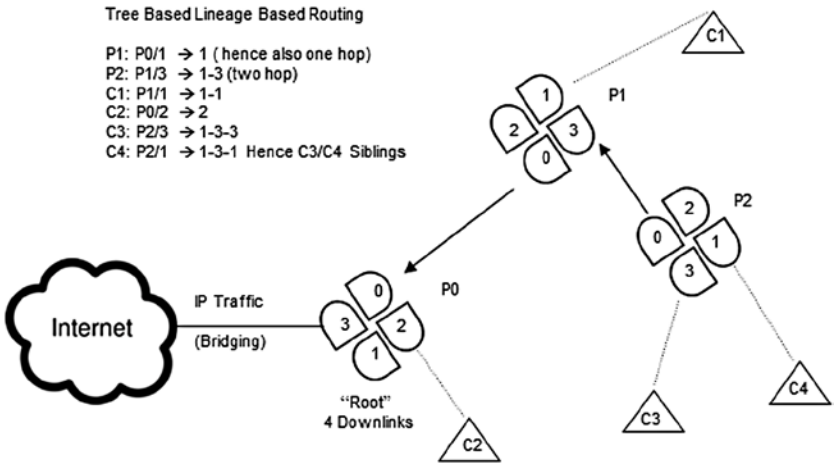

Figure 6-5. *Propagator nodes form a structured tree for networking efficiency. The propagator node with an IP connection (typically to the Internet) is designated as the "root" node. The rest organize their links accordingly, designating links as "uplinks" (toward the IP connection) and "downlinks" (away from the IP connection)*

Propagator nodes form sections of a subtree, the simplest example of which is a string of pearls (e.g., P0 ➤ P1 ➤ P2). Forming a link in the chain requires at least two interfaces: uplink and downlink transceivers. “Transceivers” here may be any form of network link: wired or wireless at a variety of speeds and with a wide variety of protocol types. Each separate transmission path or channel is a networking “slot” that may be assigned as a link in the growing topology. Generally speaking, “uplink” corresponds to moving toward the IP “root”; “downlink” is moving away.

For example, P1 slot 0 is an uplink connecting to P0 Slot 0. P2 slot 3 is a downlink providing connectivity to P2 uplink slot 0. By convention, slot 0 refers to the uplink, except for root nodes (P0). “Root” propagator nodes have only downlinks; their uplink is the IP bridge connecting either to a directly attached integrator function or (more typically) to the global Internet.

The propagator nodes in Figure 6-5 are shown with four transceivers, which could be infrared LEDs or other short-range wireless transceivers. They are placed in the general vicinity and with arbitrary orientation. Propagator nodes periodically scan the environment and reorient/reassign the slots, so there is always one uplink connecting to a parent propagator node. The choice is based on the best available effective throughput, all the way back to a root propagator node.

The parent selections are not always based on the smallest number of “hops” to the Root (a hop is counted for each propagator node in the chain). For example, P2 may be able to “see” P0, but the throughput of the direct link between P0 and P2 is inferior overall to a path from P0to P1and on to P2. In the event it was not, P2 would logically reorient its uplink, so Slot 3 would now be the Uplink facing Slot 0 of P0.

Thus, a -slot propagator node, with arbitrary orientation, may logically reassign slots 0 through 3 to ensure connectivity back to an upstream root node. The slots 0 to 3 are thus being dynamically reassigned to maintain an effective tree-based network topology.

Propagator nodes are placed in locations where they can connect to end devices and form a chain of propagator-node-to propagator-node tree branches as shown previously, all the way back to a "root" node (that bridges to IP). The primary function of the propagator nodes is to send upstream "relevant" data. In some cases, this data is being promiscuously forwarded in a public way, based simply on the arrow of transmission (toward the end device or toward the propagator node).

In other cases, the publishing agents residing within propagator nodes may be biased by integrator functions to include and exclude certain classes and categories of data based on the markers contained within the chirp streams (as described previously).

Choosing Parents Wisely

At the most basic level, propagator nodes are relays. Relays connecting to a "root" node form the branches of a tree. On power up, the primitive behavior is to become associated with a parent that provides a path upstream to the root propagator node. Generally, the closer the parent to the root propagator node, the better. The preference may therefore be, at a rudimentary level, to connect to parents with a low hop count: 0 for the root, 1 for one removed, and so on.

In a general sense, it can be expected that the bulk of Internet of Things traffic is moving mostly toward integrator functions reached via the global Internet, so there is more traffic and contention for bandwidth closer to the root propagator node. Hence, in addition to noting the candidate parent propagator nodes within its connection "zone," propagator nodes must also be able to send a "probe request" to determine the signal quality for transmission. Additionally, each device would need to know how many "sibling" propagator nodes it must contend with for access to the IP root. Siblings are additional propagator nodes linked to the same parent.

Because sibling propagator nodes are part of their own subtrees, the descendants of those siblings are also indirectly competing for the parent node's resources. In short, a tremendous amount of information must be sifted through before a propagator node selects a parent. And the situation can change unpredictably. A simplified notification of the presence to a candidate parent is required. At the base level, connected nodes transmit, through housekeeping frames, their "lineage" and "costs" of connectivity; for example:

- Name
- Current parent's name
- How many hops they are from the root node ("hop cost")
- "Toll cost" of using this propagator node (i.e., its availability)
    - Based on current processing power usage at propagator node
    - Based on number of active chirp end devices and propagator node descendants
    - Overall link quality (speed, reliability, etc.) of the path back to the root propagator node

Name-parentname-hoplevel-tollcost thus defines a broadcast beacon. Propagator node names are not globally unique; they are simply unique within a lineage subtree. Hence propagator node names, all the way up to the root, may be duplicated as long as the lineage path remains unique. Thus, two sibling propagator nodes may not share the same name, so a new propagator node with the same name as a current child propagator node will not be permitted to join that subtree.

The decision to join is then simplified to whether a prospective parent toll-cost/hop-cost ratio meets desired characteristics of current chirp packets that the prospective child propagator node would be transporting. The prospective incoming propagator node does not actually know what that data profile would be; it has not yet joined the network.

But it does potentially have access to chirp devices in its vicinity and can perform a rudimentary profile analysis, with the presumption that this is a representative sample set. Based on the profiling, if more latency is acceptable to the category of end device connected to the propagator node, a higher hop cost would be acceptable. Otherwise, a switch to a propagator node closer to the root, but at a higher toll cost, will be initiated. This is roughly an approximation of actual link quality when connected and having actual chirp devices connected to it.

It is tempting to suggest that the propagator node make a hasty connection and perhaps later, switch parents, but this is costly. Propagator nodes might then switch parents constantly, causing local oscillations (switching back and forth between subtrees), which eventually percolate to the top and decrease overall network efficiency substantially.

"Mother" propagator nodes (those with siblings) can therefore not "abandon the nest" while descendants are switching around; it would simply feed the chaos. Hence decay functions are built into the hierarchical control system that manages the network tree topology. Permission to switch parents travels at least as far up as the parent of both subtrees because both are being affected by the switch. If the child propagator nodes of those parents have settled down from a previous switch, permission is granted.

Scanning and Switching

In order to discover candidate parents, each relay propagator node must scan its environment periodically, preferably a broad scan covering multiple frequency and protocol "channels" available to the transceivers (again channels may be any type of wired or wireless link). If the propagator node has an additional dedicated scanning radio, its normal function of transporting chirps is not interrupted. Otherwise, the propagator node must request a scan "lunch break" from its parent to use its radios to scan on frequencies other than the one it is using for connection to the parent. At that point, it will need to tell its incoming link from its parent to "hold all messages." During that period, the end device clients are effectively temporarily disconnected, as shown in Figure 6-6.

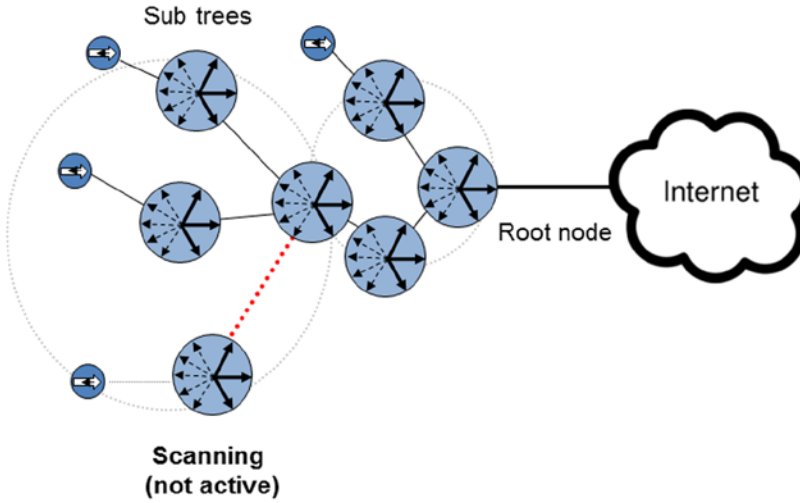

Figure 6-6. *Because each individual propagator node is responsible for finding its own best link to the IP connections, each must occasionally cease transmitting and receiving with its current "parent" to scan for alternative connections that might be of higher quality (higher speed, lower hop count, etc.). During these scanning periods, linked propagator nodes are instructed to hold traffic temporarily for the scanning node*

A parent propagator node would have multiple scan requests, which would be permitted in some weighted, round-robin manner, favoring child propagator nodes with more clients, for example. Using such a round-robin scheme, each sibling propagator node of a single parent would be granted a timed lunch break, so that no two siblings scan at the same time, thereby missing each other. The siblings may know of each other, but without mutual probe requests, they have little knowledge of the signal strength and tested link quality. Further, because the current "mother" parent's siblings (e.g., aunts) are also potential parent candidates, none of them may be in scan mode, either. Hence the scan request is being permitted by a parent's parent or grandmother. By the same token, the decision to allow a propagator node to switch is therefore also addressed by at least a grandparent to the requester propagator node.

In general, changes within a subtree (child moving from mother to aunt) will not affect the grandparent aggregated upstream throughput because both the aunt and mother are its children. So if the shift request is within the parent's siblings, the perturbation is contained and temporal. In general, it is at least grandmothers of the intended parent candidate who provide the final permissions.

For network topologies with less than a small number of hops, it is more efficient to let the root propagator node address both switch and scan requests. The root propagator node will generally have more processors and memory because it also handles the chirp-to-IP interface. As one of many "hubs" for the chirp data streams, it is also the logical place for publishing agents to reside (and perhaps collocated integrator functions).

Some of those publishing agents may want to have a say in the changes in network topology, so publishing agents may be part of the control plane managing the physical

network. Because the physical network and logical network map to each other, the only option is to change the network topology by moving propagator node connections around, based on the global (root level) toll cost/hop cost criteria. The network topology is thus managed to be in dynamic alignment with end device traffic and subscriber demand.

As with "workers" in insect colonies, the primary function of every propagator node, all the way from the edges of the network to the root propagator node, is identical. Each wants to improve its lot, but with a view to long–term network stability. This is akin to ant or bee colonies, in which the common good affects all positively. Thus a propagator node may be directed by the root propagator node to switch parents because it would streamline the small data publishing flows. Or nodes may be directed to disassociate a chirping end device and have another sibling (aunt) adopt the orphan. So each of the sibling propagator nodes may, over time, become specialist hubs for end device category clusters and the social network coalesces towards more efficient routing. This is akin to trees changing their growth to adapt to changes in sun and shade. Adaptive network trees, like natural trees, are driven by the common good of the entire tree, including all constituents, down to the chirp end devices at the edges.

Specialized and Basic Routing

Although in many cases links between propagator nodes will be IP-based, the extensible chirp protocol may also be used between propagator nodes to provide information at various levels of granularity. Within the propagator node community there may be specialist propagator nodes that will connect only to other specialist relay propagator nodes. They may limit their relay efforts to specific chirp categories or classes of end devices, thereby forming a private and exclusive logical chirp network. These specialists may use other non–specialist propagator nodes to provide the transport and routing between other specialist propagator nodes, but in effect the meaning of the data being routed is accessible only intraspecialist.

In order to support routing requests from the wider community, all propagator nodes collaborate when possible in service to the larger network. Thus basic routing is part of a common protocol and language; extensions are specialist/publishing agent–based.

The basic routing is similar to Layer 2 wired Ethernet switch stacks and their wireless mesh node equivalents (see Figure 6-7). In both cases, the tree topology ensures scalable O(n) routing overhead. In each case, there is only one uplink. The "flatness" of Layer 2-based ("switched") networks eliminates the need for additional processing and protocols required of routed networks, such as the router-based global Internet.

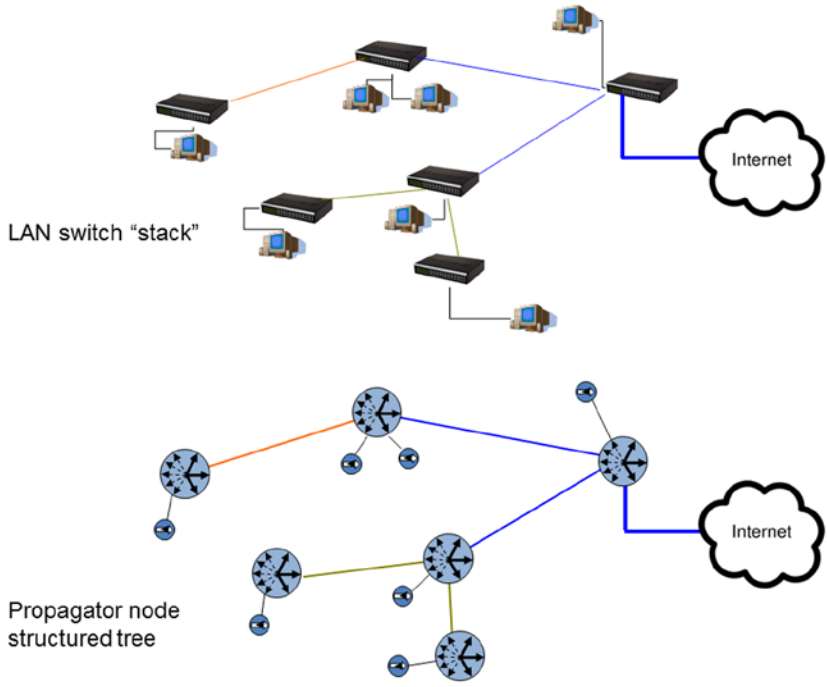

Figure 6-7. *In order to obtain reasonable routing efficiency without traditional routing overhead, the propagator node mesh is a flat structured tree, much like a "stack" of Ethernet local area network (LAN) switches. But unlike the LAN switch stack, distributed intelligence in each propagator node manages uplinks and downlinks to avoid loops while also maintaining alternative paths to allow for rerouting around failures*

Housekeeping Frames for Network Intelligence

As noted earlier, the very basic "housekeeping" information that relay propagator nodes may transmit must minimally include the hop cost, toll cost, and parent name. The parent name is needed so that a prospective child can talk directly to the parent. Recall that the grandparent manages scan and switch events, so it knows whether a better parent is available, but is out to lunch performing a scan. A propagator node might be left awaiting association permission from a prospective parent node's parent until a scan is over. This delay ensures that after connections are made, they do not have to switch to a better candidate that is discovered after some later scans. The grandparent is being proactive.

Thus, the very basic housekeeping frame from all propagator nodes must contain:

- My name
- My parent name
- My hop level (from the root node)
- My toll cost

Newly powered–up or unconnected "orphan" propagator nodes send and receive probe requests from multiple connected nodes in their vicinity during their scanning period after power up. From these, the "orphan" propagator node can surmise which candidates are siblings, based on their parent name. Should it join any sibling, it is assured of collaborative alternatives within the same subtree (the aunts). This engenders its "survival" in terms of redundant paths with minimal changes; the rest of the subtree back to the root propagator node would be unchanged for between-sibling switches. Routing updates are needed only at the last hop. In contrast, switching between entire subtrees is more onerous, especially if that subtree's siblings are not available as backups. Survival favors joining subtrees with multiple accessible sibling mother/aunts.

Latency and Throughput Tradeoffs

Exchange of housekeeping frames enabled orphans to discover the presence of potential parents. Potential parents' relative proximity is measured during probe requests to determine effective link quality, which is included with topology analysis in the selection process. The total available throughput in a string-of-pearls link is simply that of the weakest link–the link with the worst "performance".

Candidate parents may thus receive pings to test aggregate link quality all the way up to the root propagator node. In general, each propagator node has an inherent predilection to choose the best "lineage" to connect with. But there are tradeoffs. Ideally, all things being equal, propagator nodes would want to connect as close to root propagator nodes as possible because Internet of Things traffic is largely upstream. However, the link quality of a direct wireless connection to a lower-hop-count but physically more-distant propagator node may be much worse than routing through more intermediary propagator nodes.

In the previous examples, overall back haul throughput from all upstream traffic to the root degrades as the tree topology is modified by toll-cost and hop-cost ratios favoring low hop cost (as shown on the right of Figure 6-8). But when the toll cost of lower-speed links is considered, the topology at left in Figure 6-8 is actually more efficient overall.

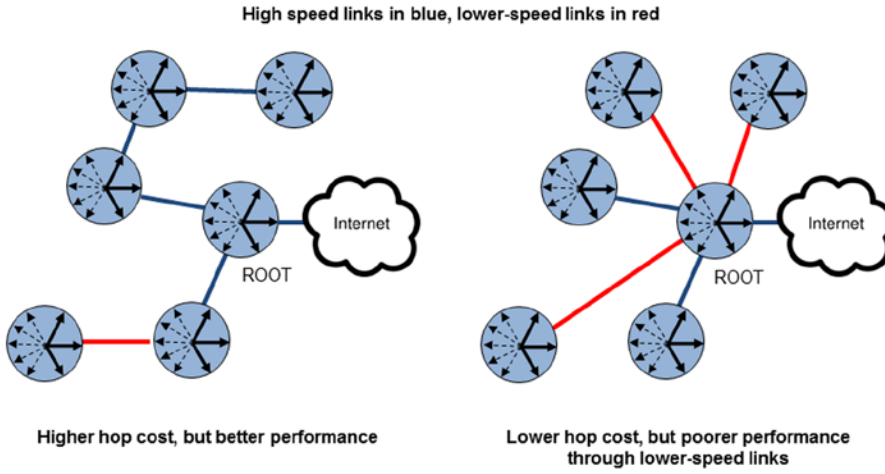

Figure 6-8. *Although lower hop cost (fewer node-to-node links) is a first-order approximation of best overall performance (on the right), sometimes lower speed link paths (due to distance, perhaps) make a higher-hop-count topology more efficient (on the left)*

In addition to overall link quality ascertained through pings, the availability of the candidate parent to service additional requests drives the final decision. Great overall backhaul throughput is academic if the node is already saturated, based on its limited processing power. Toll cost provides information to the nodes regarding levels of availability. Higher-toll-cost propagator nodes are being selective, mindful of their own limitations and therefore protecting their existing clients from being crowded out. Thus allegiances are formed, wherein propagator nodes develop preferences to belong to particular subtrees that demonstrate healthy characteristics (e.g., multiple sibling accessibility, etc.)

Routing Table Updates

Having joined the network, propagator nodes must now begin relaying chirp broadcasts in their vicinity. Propagator nodes would typically have one uplink to maintain the tree structure, although multiple uplinks servicing disparate trees (to avoid cycles) are also permitted. Multiple downlinks can service both chirp and IP traffic on both the same and distinct wireless interfaces. The uplinks could be either chirp- or IP-based (e.g., Wi-Fi or Ethernet).

For each disparate uplink, routing tables are maintained that provide Layer 2 switching network functionality. Packets are moved either up the tree or down the tree via the immediate children, along the tree branches. The decision is based on a condensed routing table, updated by each propagator node, based on a full housekeeping frame sent periodically and circulated within the relevant subtree.

Each housekeeping frame is tagged with a counter number. Because housekeeping frames will travel multiple paths in broadcasts, propagator nodes ensure that the same counter number is not rebroadcast. Further, each propagator node (and its publishing agent, if so equipped) may decide how far up or down the tree to provide the broadcast. For example, a parent switch to sibling aunt requires no further broadcasts than the last hop routing table.

Eventually, each propagator node is aware of the following:

- Its own immediate children
- In the case of relay propagator nodes, their children, and so on
- Adjacent propagator nodes that may serve as alternate parents
- Its current overall current link quality and throughput
- Through scanning, the overall link quality of alternate parents

Over time, the cost of switching back and forth is reduced by developing more data on the current parent and on its alternatives. This information leads to stable networks at the local levels.

The routing table is available to all members of the current subtree to (at least) the level of a grandparent. Each propagator node is aware of is entire subtree of descendants below it, at least two hops down. After that, the knowledge is somewhat irrelevant because its grandchildren, on having the packets delivered to them, will know how to relay them further. All the grandparent needs to know is *roughly* where the chirp parent propagator node resides—the portion of its descendant subtree (a general direction of routing suffices). If chirp devices move around, one or two packets intended for them will be lost (recall that there is no retry or retransmission in chirps). For each chirp descendant under its care, propagator nodes need to be aware only of the following:

- The chirp device descendant's immediate parent propagator node
- The location of that parent propagator node within its subtree (e.g., lineage)
- That the lineage path from the root propagator node to chirp device exists

Some chirps will be picked up by multiple propagator nodes, and each will rebroadcast the packets in the directions specified by the arrow of transmission. However, in each case it will tag the packet with the chirp device's immediate propagator node, which is the last part of a lineage tree. Multiple chirp packets will thus travel separately upstream through different relay paths, from multiple propagator nodes that pick up the chirps in their vicinity. Multiple lineage paths are available.

Multiple paths are useful when redundancy is desirable. Such is *not the case* with chirp sensor data (given the relative unimportance of any single chirp), so pruning of multiple paths is performed at the grandparent level. Chirp packets are relayed through one node only, typically the node closest to it and therefore the best link quality. Others, also picking up the chirps directly from the devices, are directed to ignore those chirps. The chirp device is now assigned a unique lineage or relay path back to the root. Thus, even in the case of unidirectional chirp streams, an association is made to prune redundant traffic.

The Power of Local Agents and Integrator Functions

When local publishing agents are added to propagator nodes, a tree-based, scalable, hierarchy-driven control system emerges. Filters are applied to progressively reduce redundant data upstream and to define preferred routes or destinations. Here is the beginning of small data flows, as chirp data being sent upstream continues to be more refined as it passes through multiple rule–based logical sieves.

As streams of chirps travel upstream toward the IP root in this model of the Internet of Things, agents within propagator nodes at strategic (often branching) locations along the route may perform local pruning, aggregation, and exception handling, thereby reducing the traffic and improving load performance. Because multiple agents can be operating on the same data, some form of collaborative scheduling and sharing of timing requirements is needed.

Task Scheduling within the Internet of Things

In the emerging Internet of Things' three-tiered architecture, propagator nodes manage the flow of aggregated and pruned data between end devices and integrator functions. When these propagator nodes incorporate a publishing agent (and the requisite IP interface), they may have access to two vital pieces of information supplied by integrator functions that are the receivers in the publish/subscribe framework. This information includes the following:

- **For routing:** The location of the integrator function that is in search of data characterized by its specific category or originating location (the publish/subscribe "neighborhood" described in Chapter 5)
- **For scheduling priority:** Timing requirements for delivery of the data (outdated data may have no value and need not be propagated through the network), along with estimates of time required for data to reach integrator functions

Communication between integrator functions and publishing agents takes place over the IP interface using standard Representational State Transfer (RESTful) or Simple Object Access Protocol (SOAP) protocols.

Through their exchange of housekeeping frames and observed traffic routing delays, propagator nodes are aware of how much time is taken moving packets across their managed chirp network, across the chirp-to-IP bridge. On the chirp side, latency is more deterministic: a simple count of the number of hops to the root node defines, in large part, the delay. On the IP side, things are more complex because the IP highway is being shared by other devices from other communities.

Propagator nodes are, however, in periodic communication with destination IP addresses. Simple ACK protocols within the RESTful API can provide current or predicted estimates of IP traffic. Working backward, propagator nodes back-calculate when chirp bus loads should leave. This feeds the collaborative scheduling and stack management

routines. The scheduler may also drive aggregation (bulking) to ensure an equitable compromise between bus size, its frequency, and the IP cost at different times.

Smaller bus loads will leave more frequently for passengers in a rush; others will be compensated by a lower bus price for travelling on larger but less frequent departures. Some buses may arrive earlier, others later, but the schedule stacking is usually managed proactively. Supply and demand of the chirp packets and their arrival is driven by dynamic subscriber demands. This is a dynamic form of prioritizing, based on chirp useful life and subscriber demand information generated by the integrator function(s).

Higher-level Interchange

Similar standard IP-based protocols are used between integrator functions and filter gateways, as well as to create relationships among two or more integrator functions as neighborhoods and affinities are created, modified, and abandoned over time based on known and discovered data sources.

As noted in Chapter 5, integrator functions may be collocated with propagator nodes in some applications. These distributed integrator functions may be relatively simple, but their strength lies in numbers and their ability to support multiple interpretations of the same data. Distributed integrator functions in this scenario initiate corresponding actions with significantly lower latency than if everything were sent round-trip to distant integrator functions. There are situations in which chirp data needs must propagate all the way up, but like the Mars Rover, when latency matters, some level of local autonomy is essential to the survival of a network burdened at the edges.

THE POWER OF PUBLISHING AGENTS

An example illustrates the savings in IP traffic and improvement in responsiveness for a 100-sensor network. Consider, for simplicity, a ten-node string of pearls chain, with each relay propagator node supporting ten sensors, all the way back up to the root. For example, these sensors could be part of an underground coal mining tunnel, with propagator nodes forming the lifeline for both IP and chirp traffic.

Simple rule-based logic in distributed integrator functions watches the methane gas occurrence across the tunnel. The development of methane in one region could also affect adjacent regions, so a blob of methane gas publishers may appear abruptly and unexpectedly.

Sending such “exception handling” upstream to big data servers is clearly valuable. It is questionable whether routine and acceptable readings would be transmitted. But without some local handling, there may be no way of defining an exception, without a base line of routine readings. Hence publishing agents may also maintain some short history.

In lower-end and consumer versions of propagator nodes, there would be limited agents available—most data may be pushed upstream to parent propagator nodes and on to separate integrator functions. But multihop paths and their associated

latency may be unacceptable for some mission-critical enterprise applications. In a previous era, Programmable Logic Controllers (PLCs) wired to sensors and actuators on the factory floor, managed the deluge of real-time, latency-sensitive machine-to-machine traffic, escalating only that which fell out of their rule-based relay ladder logic diagrams used by PLCs. Today, that same approach can be applied to rule-based agents residing on propagator nodes, close to the sensor/actuators. This reduces latency for enterprise class machine-to-machine communications.

Managing Multiple Isochronous Relationships

As introduced in Chapter 5, regardless of whether device communication is IP- or chirp-based, independent control loops (with publishing agents as intermediaries acting as the translation mechanisms between the upper and lower control loops) are inherently more efficient than round-tripping. Some devices, such as smartphones, are inherently chirp-capable (e.g., IR and Wi-Fi) and can participate in both control loops, acting like a bridge between the two banks of the river, each with its own control loops.

Beyond round-trip latency considerations, there is a more fundamental reason for this tiered control and communications model. The language and vocabulary of end devices is fundamentally divergent from that at the big data server level. Sensors publish their limited view of the world, whereas big data provides insights into a more comprehensive world view, incorporating multiple sensor streams, past history, future trends, and so on. Because function dictates language and vocabulary, some form of translation is required—one cannot expect purpose-built machinery to communicate directly without translation.

In the contemporary IP-based thin-client model, any translation of data to a format palatable to big data consumers must take place *before* sensor data enters the IP network. Needless to say, that puts the onus on the end devices and their machine-to-machine communication protocols to be intelligible. What was a terse, purpose-built dialect now has to be interpreted in a device-abstracted language. Agents and their location within the local control loop reduce this burden, as shown in Figure 6-2.

An Organic Solution for the IoT

One cannot always know, *a priori*, the type of categories of interest for specific integrator functions, any more than winds can always provide focused beams of pollen to their awaiting subscribers. Some discovery is needed, though; at the very least, notification from “mother” propagator nodes that a new category of sensor chirps has become active in a geo location under its care (e.g., a subtree of the network). Notification summaries of sensor activity would therefore be periodically provided. Interested integrator function subscribers can then direct their publishing agents within propagator nodes (if available) to provide the level of granularity/aggregation/pruning/exception handling needed to optimize data gathering.

Over time, an agent-based, machine-to-machine social network emerges, tapping into the full richness of data offered by the Internet of Things.