CHAPTER 5

Small Data, Big Data, and Human Interaction

Integrator functions are the location in the Internet of Things (IoT) where the chirps from hundreds to millions of end devices are analyzed and acted upon. Integrator functions also send their own chirps to get information or to set values at devices—of course, these chirps' transmission arrow (refer to Chapter 3) is pointed *toward* devices. Integrator functions may also incorporate a variety of *external* inputs, from big data to social networking trends to weather reports.

Integrator functions serve as the human interface to the IoT. As such, they will be designed to reduce the unfathomably large amounts of data collected over a period of time to a simpler set of alarms, exceptions, and other reports for consumption by humans (or computers). In the other direction, they will be used to manage the IoT by biasing agents within propagator nodes (refer to Chapter 3) and other devices to operate within certain desired parameters.

In this way, integrator functions create the publish/subscribe network that extracts meaning from the Internet of Things. Integrator functions define neighborhoods and affinities that are the key relationships within the IoT, regardless of geographical location or network topologies (see Figure 5-1).

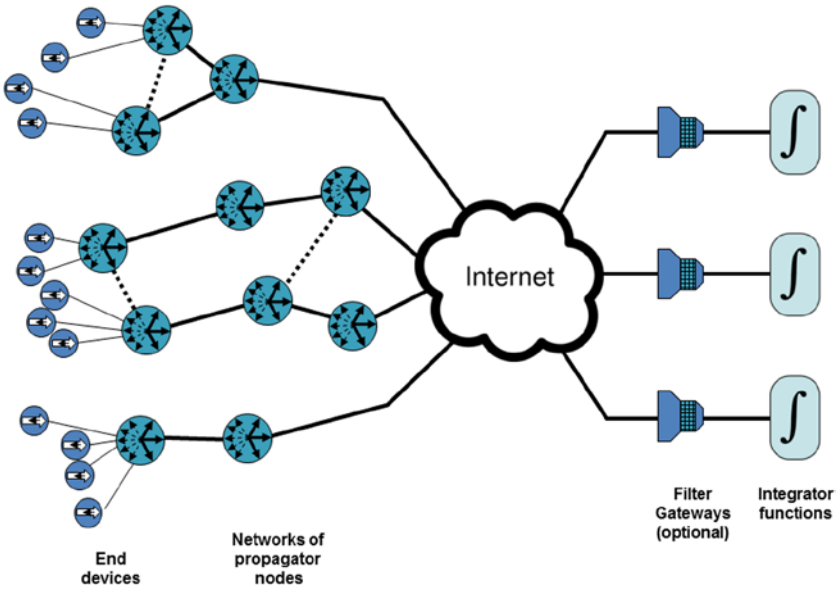

Figure 5-1. *Integrator functions are logically at the center of the Internet of Things publish/subscribe network, although their physical location is unrestricted*

Using simple concepts such as "cluster" and "avoid," integrated scheduling and decision-making processes within the integrator functions will allow much of the IoT to operate transparently and without human intervention. Only a single integrator function might be needed for an average household, operating on a smartphone, computer, or home entertainment device. Or the integrator function could be distributed and scaled up on racks of far-flung processors for a huge global enterprise, tracking and managing energy usage across a corporation, for example.

The "Brains" of the IoT

The most typical form of integrator function consists of specialized software operating on standard off-the-shelf computing platforms. Requirements for the integrator function are for the most part similar to those of other computing-intensive applications: processor horsepower and memory.

For maximum economy of scale and full exploitation of Moore's Law over time, widely deployed computing platforms and operating systems will likely be the primary targets for integrator function software development. Computing power and memory will be commensurate with the amount of data to be analyzed and/or the quantity and sophistication of the end devices to be controlled. Low-end home automation may be achieved with a smartphone and an appropriate app, while monitoring an extensive global process control enterprise (such as oil production) might require clusters of high-end processors with redundancy and fail-over capabilities (see Figure 5-2.).

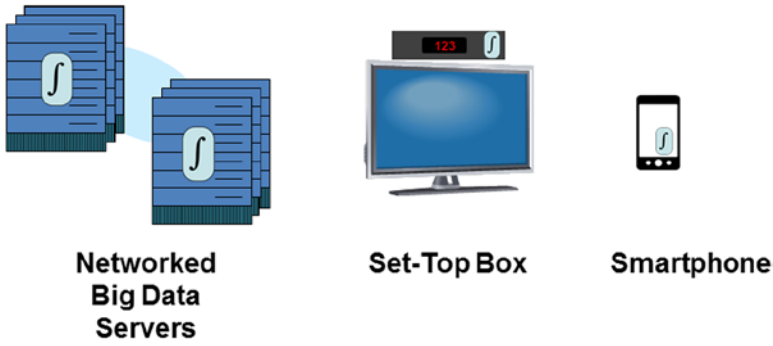

Figure 5-2. *Integrator functions may be hosted on a very wide range of general-purpose and broadly deployed computers and devices*

Fortunately, the processor and software development paths (clustered processors, Apache Hadoop distributed file systems, converged network adapters, solid-state storage, etc.) that support expanding big data applications will seamlessly incorporate the emerging Internet of Things integrator function as well.

For Once, IP Makes Sense

To minimize cost and complexity, chirp protocols have been described in the preceding chapters for the end devices that will form the vast numerical majority of the Internet of Things. As explained, these simple protocols will not be sufficient (or even formatted) for transport across the global Internet. Instead, data streams to and from end devices will pass through one or more propagator nodes before routing via standard IP over the Internet (or rarely, a private IP network or VPN) to one or more integrator functions.

The logic in requiring an IP-capable propagator node in the data path now becomes obvious: at the lower networking protocol levels, integrator functions may simply rely on standard IP networking capabilities *already* deployed in typical operating systems with a simple connection made to the Internet via Gigabit Ethernet or other existing and widely deployed interfaces. The architecture of the integrator function builds on many of the same principles as cloud-based computing and will benefit from investments and developers in cloud-based servers and Internet backbone build-outs.

IP–based integrator functions also directly incorporate legacy Internet of Things devices that offer only IP interfaces. This creates an easy transition to the emerging IoT architecture for millions of already installed sensors and actuators, as well as for higher–performance end devices that will remain IP. Integrator functions can also interact directly with millions of existing web-based data feeds and services, creating richer meaning when these sources are combined with IoT data streams.

The downside to this approach of leveraging the global Internet and commercial systems is that very large data streams and busy network interfaces could bog down a general-purpose processor. For this reason, filter gateways (see following) may often be deployed as a specialized appliance to forward only meaningful data (as determined by the integrator function). This ensures that the computing resources of the integrator function may be focused purely on analysis and control tasks.

Extracting the Streams

But as described in preceding chapters and more fully in Chapter 6, the majority of Internet of Things data bundled and forwarded by propagator nodes consists of a distilled stream of chirps encapsulated in IP, not wasteful discrete IP packets for each end device (see Figure 5-3). An internal gateway process within the integrator function must unpack and identify chirp streams for action.

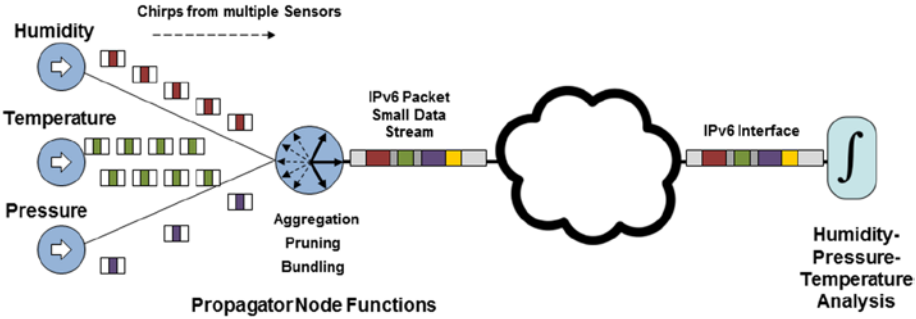

Figure 5-3. *Chirps from IoT end devices are aggregated, pruned, and bundled in the propagator node network; then encapsulated in IP for delivery to an integrator function as a small data "stream" from which the data may be analyzed. A similar process operates in reverse to deliver data to end devices*

Similarly, for outbound traffic such as control packets to valves in a process control application, the integrator function will package chirps within IP packets in a form understandable by the propagator node network. Along their path, these packets will be disassembled, reassembled, repeated, and pruned as necessary to reach the target end devices.

Analysis and Control

With the various inbound streams identified and separated, the integrator function may begin forming a "picture" of the *neighborhood* in which it is interested. Neighborhoods are described more fully later, but they basically consist of a universe of devices that correspond to the type, location, activity level, and so on that the integrator function has been programmed to seek out. For inbound streams, the end devices are *publishing* data to which the integrator function *subscribes*. (And the inverse is true for end devices controlled by the integrator function.)

But the integrator function is not *necessarily* limited to a hard-coded set of specific locations or device types in forming the publish/subscribe neighborhood. A broader set of possible relationships, called *affinities*, may allow an integrator function to create a neighborhood from unrelated end device streams if an interesting or recurring pattern is noted among devices (see Figure 5-4).

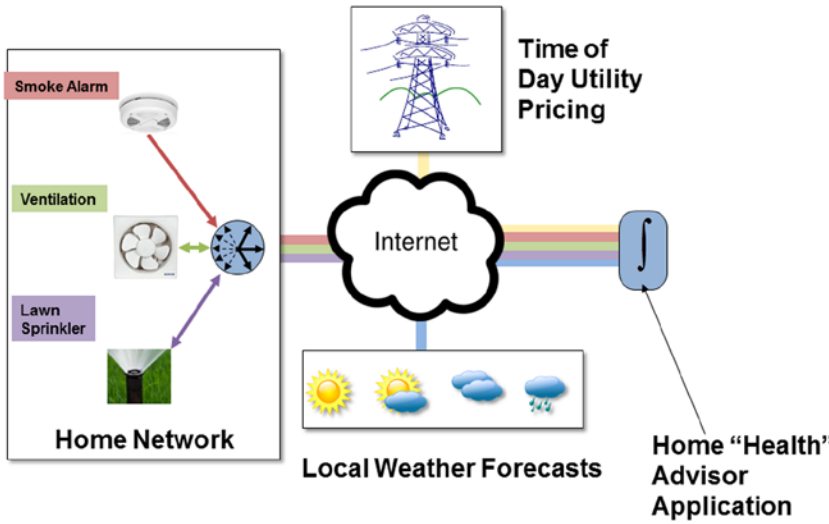

Figure 5-4. *Integrator functions subscribe to interesting data streams published by IoT end devices. An integrator function's "neighborhood" may span the globe. "Affinities" with other potentially associated data may also be exploited to create richer information*

This capability begins to tap into the tremendous potential of the Internet of Things to create useful information and meaning by collecting information from a wide array of devices, many of which may have been deployed *by other entities for other purposes*. To fully appreciate the potential, it is necessary to look beyond traditional end-to-end networks and even software-defined networking concepts to examine the development of meaning from a wide array of sources. One way to appreciate this concept is through a neighborhood analogy.

Chirps to "Small Data" to Big Data: An Example

The build-up of data from many devices is similar in some ways to the build-up of musical tunes from discrete notes. Individually, an isolated note offers no musical information (emotional content, beauty, etc. as determined by a listener). But a series of notes from many sources (the instruments in a symphony, for example) form tunes that human listeners may interpret.

Similarly, chirp sequences (in parallel or serial flow) form "tunes" after a fashion. The chirp stream "tune" is used as a signature pattern or a data payload. Or a concatenated and encrypted version of both (see Chapter 6) where encryption includes delayed transmission as in syncopation. Multiple tunes are really a jumbled version of hidden information, where even the silence may have meaning, known only to intended receivers.

Although humans can hear a birdsong, the chirp sequence "meaning" is known only to the birds. Although humans hear the tune, they cannot decipher it (see Figure 5-5). Birdsong signatures ("blue jay") and payload ("intruder") are both tunes, so it is unclear where one sort of tune melds into another. Hence, humans can hear all the myriad bird conversations in the park and yet understand none—they do not have a decoder key.

Figure 5-5. *The movement of the neighborhood cat sets off "alarms" in a number of "sensor devices" (birds). A human observer may correlate information from multiple senses and understand what is taking place*

Bird chirps respond to changes in the environment. For example, a cat walks through the park. Human eyes can follow it, noticing how the chirps follow the cat's motion as it moves from one tree to another. Chirp tunes will change both in the sequence of tones and their intensity. An observer may be able to discern activities common to the same consensual domain by matching patterns in two *different* sensor domains (eyes and ears) and "putting two and two together." Multiple sensor fusion (eyes and ears, in this case) drives the human inference engine.

Over the course of a month, the cat may visit different parts of the neighborhood. Although there may be trends to these movements, the sampling duration may need to be months to accurately pinpoint "affected" regions. The quantity of data to be analyzed is considerable. Some may need to be stored and reviewed later by the big data analysis engines that are predicting trends based on past history.

Over time, it is noted that this "small" data pattern repeats itself around dusk most nights. "Big" data engines may then infer that a nocturnal animal (e.g., a cat) is causing a "disturbance" in the "reference" signal. Individual chirps and even the combination of chirp streams that create "small" data are unintelligible in isolation. But they may be processed into a more coherent form, which in turn is used to draw conclusions about the environment *not transmitted per se* in each "small" data transmission.

Putting small events together to infer a complex event or trend is difficult, whether in the natural world or the Internet of Things. It may require a control system component, Bayesian reasoning, to filter out the noise from reference signal disruption. "Small" data events, based on observation, propagate "up" for "big" data analysis and action. An immense number of small events feed myriad chirps that may be integrated into complex event analysis.

This example has described only one sort of event (birdsong) in one neighborhood. But as seen in the following section, additional richness in analysis comes when integrator nodes expand on the concept of neighborhoods by actively seeking out and incorporating affinities.

Neighborhoods and Affinities

Internet of Things neighborhoods may be thought of as interesting aggregations of data sources that may be examined and collected by an integrator function. Locating and subscribing to a particular chirp stream may be directed by human programming (e.g., "monitor all moisture sensors in agricultural fields in these four counties").

In this case, the neighborhood is defined geographically, so the integrator function may seek out interesting data streams from many candidates by searching for a particular "signature" of device type (from markers in the chirp packet; see Chapter 6) and location information appended by propagator nodes. Subscribing to these streams allows the integrator function to build up not only a snapshot of current conditions but also to observe changes over time. This data may then be used to generate reports or alarms as needed for human observation.

But the preceding example is not much different from a point-to-point IP data stream type of relationship. In fact, IoT neighborhoods *need not* be bounded by geography, end device type, or any other characteristic. Nor need they be preset by human operators. Instead, they may collect chirps across a wide spectrum of small data flows.

By subscribing to soil moisture sensors, temperature gauges, weather reports, reservoir levels, electric utility time-of-day rates, video images of crop height and ripeness, and so on, it might be possible to create a model that will allow the most cost-effective and timely irrigation of fields, for example. This could be effected either by outputting a report to a human field hand, or the integrator function might simply open the correct valves for the precise time needed (see Figure 2-8).

Independent but *interacting* elements of the real world, each of which is represented by data flows and sources (whether from Internet of Things end devices, the global Internet, or another source) represent affinities of data. These discovered affinities of data may prove to be much more powerful than human programmers might predict in advance. For that reason, it is fundamental that the underlying architecture of the IoT integrator function software allows for independent searching out of potentially interesting data sources by intelligence operating within the integrator functions. (The mechanics of this affinity-seeking intelligence is more fully explored in Chapter 6.)

Note that not every deployed integrator function will incorporate this independent data-seeking capability. In many cases, the role of the integrator function will be more narrowly defined to a specific application or locale, partly for cost and control factors, but also to allow the use of cheaper computing platforms (owing to the need to analyze less data).

Public, Private, and Some of Each

The broad architectural definition of the integrator function and its varied application uses mean that there will be different kinds of neighborhoods formed. This is enabled by the incorporation of public and private markers in the structure of the chirps themselves (see Chapter 6). As in the previous birdsong example, the chirp structure contains both

addressing and payload information in a form that will be unintelligible without the proper "key."

Somewhat akin to a Virtual Private Network (VPN) in traditional IP networking, chirps with private markers may traverse a broad network alongside chirps with public markers. In the integrator function publish/subscribe model, security is provided at the end point only. As in trees, where a particular pollen particle may be propagated in any direction, but only a receiver (the flower) programmed to "receive" the message will act upon it, so chirps with a private marker will be inert to any but the desired integrator function.

These private chirps might be seen in Original Equipment Manufacturer (OEM) environments, in which, for example, an application that monitors diesel generators and schedules maintenance as required (low fuel, hot bearing, clogged filter, etc.) might be offered for a specific manufacturer and then only for those units under warranty.

Other chirps (likely the statistical majority) will be public, available for inspection by any "interested" integrator function that builds the chirp stream into a neighborhood. As with emerging social networking norms, in which a wide variety of information is made publicly available by individuals, it is likely that some entities deploying some types of end devices will use public markers only, making those chirp streams available to any integrator function that detects it and subscribes. Again, subscription is an activity of the integrator function only; not of the end device.

It is likely that the some of the most interesting and powerful big data applications of the Internet of Things will come through some combination of public and private chirp streams and small data flows (see Figure 5-6). So hybrid environments with private and public chirp streams sharing portions of propagator networks will be quite common.

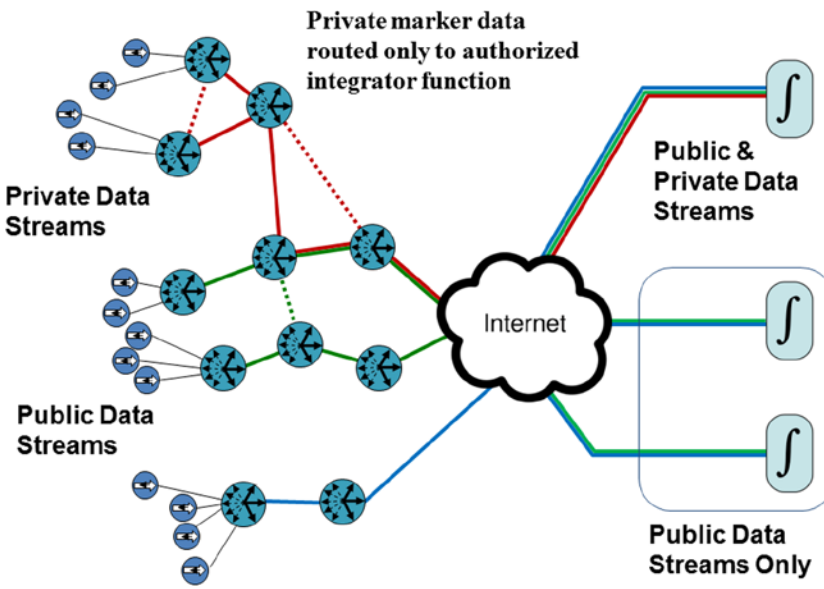

Figure 5-6. *Although some proprietary applications will use private markers to restrict use of chirp data, most chirp streams will be fully public for analysis by any "interested" integrator function*

Bias Bonus

The potential power of noncontiguous information neighborhoods formed though integrator node affinities selecting among millions of chirp streams is enticing. But seeking out specific chirp streams from desired devices in the cacophony of the Internet of Things will also be important. Especially for OEM and proprietary networks that go beyond generic functions, some method of network tuning may be helpful.

As introduced in Chapter 4, one class of propagator nodes will be equipped with an internal publishing agent that is accessible to one or more integrator nodes. These publishing agents may interact with complementary processes in the integrator function to cause the propagator nodes to preferentially forward some types of packets, perform proprietary chirp bundling, spoof repeated chirps (or refrain from spoofing), and perform other roles on behalf of the integrator function.

In this fashion, integrator functions may construct preferred networks, optimize data flows, and seek out specific types and locations of end devices. The publishing agent–integrator function interaction my again be proprietary or more open. The proprietary interaction is relatively straightforward because there is (by definition) an IP-based connection between integrator function and propagator node. This will allow for secure tuning of data flows between propagator node and integrator function, in a manner similar to a *software-defined network*.

But in a pubic (or mixed) environment, the situation is more complex. Because propagator nodes may be servicing multiple chirp streams, bias must be defined in a way that is neither proprietary nor permanent. The effect of an integrator function on such a (non-private) publishing agent will ebb and flow in an organic way. Repeated interactions with a particular integrator function will bias the propagator more heavily in favor of that integrator functions publish/subscribe requests, as seen in Figure 5-7.

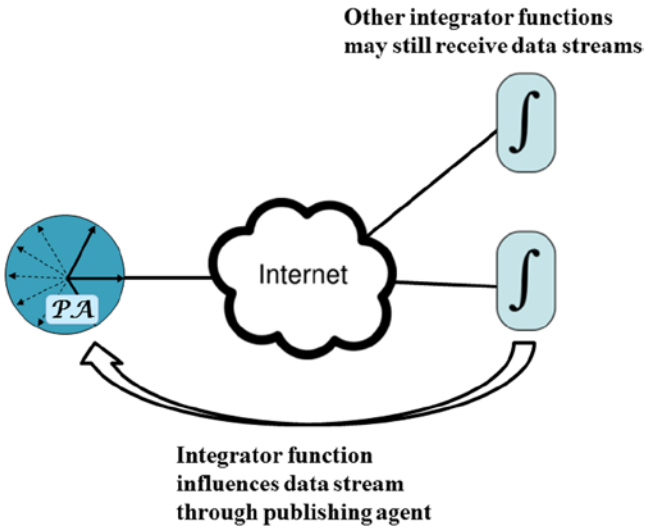

Figure 5-7. *Some classes of propagator nodes will contain publishing agents. The agents interact with integrator functions to "tune" the data forwarding. The bias of the propagator node may be reconfigured over time and fade out if not reinforced by the integrator function. Publishing agents also advertise the availability of potentially interesting new end devices for possible subscription by the associated integrator function*

But if, over time, that interaction ceases or is reduced in frequency, the propagator node will revert to a more promiscuous (nonbiased) forwarding model or will respond to a different integrator function that shows more "interest" by more active biasing. The weights and dwell times of this biasing may be configurable at the propagator node.

Searching for and Managing Agents

Publishing agent–equipped propagator nodes will typically advertise summaries of attached end-device types (including those farther down the "tree" accessed via other propagator nodes), locations, and other characteristics. Integrator functions may use this information to identify publishing agents that have offered interesting chirp streams. (There will also be secure proprietary modes in which propagator nodes do not advertise available data types; instead being specifically targeted by integrator function programming.)

The integrator function establishes a connection to one or (often) many of the publishing agent(s) in order to read and then set certain parameters. It is likely that these publishing-agent-equipped propagator nodes will be placed at logical "junctions" for "branches" of the Internet of Things, creating useful points for management and control. Because publishing agents always reside in propagator nodes that are equipped with an IP gateway, standard IP protocols will be a straightforward medium for their interaction with integrator functions.

This integrator function bias will include settings regarding frequency of transmission for target chirp streams (every chirp, periodic, only when state changes, etc.), exception handling, multicast bundling/pruning, and so on. Propagator nodes will announce discovery of new candidate chirp streams for potential inclusion in preferred forwarding lists. Integrator functions will also restrict the forwarding of some chirp streams to limit the proliferation of unneeded or redundant data.

The biasing of a publishing agent by a particular integrator function is not permanent; over time, requests by other integrator functions may take precedence if there is not "reinforcement" by the originally requesting integrator function. This will allow for organic reconfiguration of the network due to changing needs, seasonality, and other factors.

High- and Low-Level "Loops"

An interesting byproduct of this architecture is that there will essentially be two networking "loops" operating in the network when publishing agents are present in the propagator nodes and are biased by integrator functions.

Propagator-node-based processing for end devices, operating closer to the devices, provides a more equitable distribution of resources. Integrator functions are thus freed from handling communications chores for thousands of end devices.

That more mundane work of pruning and aggregation is then delegated to publish/subscribe agents within the propagator nodes, closer to the end devices. The *control loop* is then effectively split into two isochronous control loops: one loop between the end devices and the biased publish/subscribe agents within propagator nodes, and the second between those agents and their associated subscribing integrator functions.

In the traditional IP–based thin client model, there was effectively one control loop between devices and servers, so end-to-end delays, error checking and correction, and so on are necessary (not to mention a costly full IP stack in each device, as discussed previously). But with the agent located within the propagator node, end devices may continue to converse in simple terse chirps. The end device chirp stream is being converted into a small data flow, to which integrator functions may subscribe. The overall architecture is more scalable and more efficient by *disassociating* the two control loops, as seen in Figure 5-8.

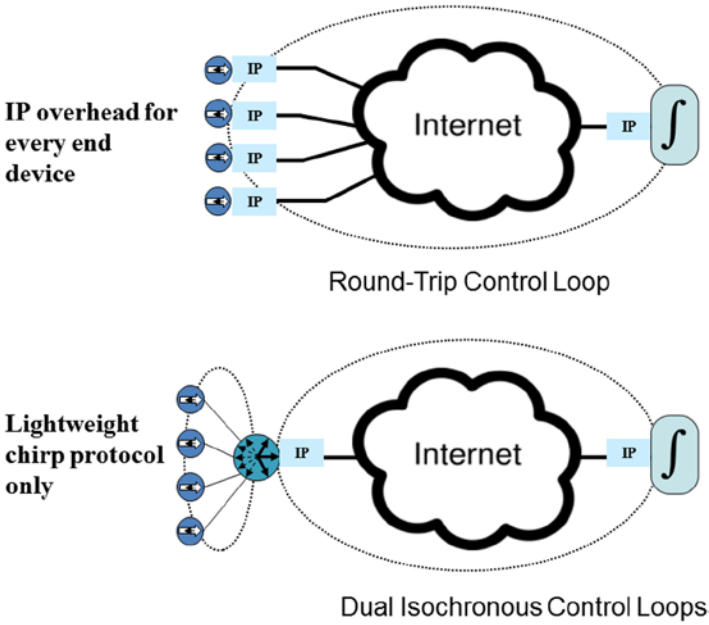

Figure 5-8. *Traditional IP networking models extend the control loop end-to-end, demanding deterministic performance and burdening end devices with expensive processor power and memory. The emerging Internet of Things architecture creates separate control loops, allowing minimal networking investment at the end device and providing better local control without the delays of round-tripping*

In this distributed and balanced setting, the publishing agent within the local propagator node acts as an extension of the integrator function, managing the exceptions that interest them: the higher-level loop. The task of pruning and aggregating is delegated to a lower level of control. Round-tripping is obviated.

Using the Mars Rover as an analogy, Mission Control is kept abreast of "interesting" developments, but local control of sensors/actuators is handled autonomously by resident software agents. This obviates needless round-tripping between the rover and earth, providing a more equitable distribution of tasks and resources. This is more efficient because it also reduces both traffic and server load. The output from biased publishing agents is a more palatable edited small data flow.

Regardless of whether device communication is IP- or chirp-based, a layered control loop (with agents as intermediaries acting as the translation mechanisms between the upper and lower control loops) is inherently more efficient than round-tripping.

By contrast, in the traditional IP thin client model, that translation would take place in the cloud, demanding that data originate from end devices in a format palatable to big data consumers. Agents and their location within the lower control loop reduce this burden on the end devices of the IoT.

Agents are bilingual by design. End device-to-publishing agent conversations can be in one language (chirps), more suitable for lower-level conversations. The sensor-motor control loop of the Mars Rover involves a different vocabulary than the command control

loop with Mission Control. Local software enables sensors and actuators to be in a close, tight control loop and to do what they were designed for. Other software, with access to this lower control layer, provides Mission Control with the level of granularity needed.

An intermediate agent-based architecture is also closer to the publish/subscribe frameworks that big data systems are familiar with, and so may allow for easy extension to the Internet of Things integrator function architectures. Through web services, cloud servers subscribe to multiple sources of data. Big data systems may be viewed as marketplaces in which publishers/subscribers or data providers and consumers meet and exchange. The "exchange" is one service that enterprise middleware software provides at Layers 7 and above on the network stack. For example, Tibco (www.tibco.com) provides a platform in which real-time feeds are both published and consumed. Multiple and diverse applications employ generic and extensible real-time publish/subscribe "exchange" infrastructure to conduct business. The existence of these models should make the incorporation of integrator function data very straightforward.

Human Interface and Control Points

In the Internet of Things, the integrator function collects the small data flows that develop from combining chirp streams. Somewhat like the human observer in the earlier birdsong example, the integrator function may correlate events and observe patterns from millions of chirps that would be unintelligible (individually or en masse) to a human observer.

Thus the integrator function is the point at which data may be turned into information for consumption by humans. Reports may be generated which highlight conditions in the field, thresholds for certain events calculated and alarms posted, and so on.

For example, a power plant (see Figure 5-9) might monitor thousands of points for temperature variation, vibration, fluid leakage, and other factors. An integrator function would not only monitor individual sensors for out-of-tolerance values but might also examine the interaction of changing values across multiple types of sensors deployed on a variety of equipment. Does an increase in temperature and vibration at a number of related locations represent a potential trouble spot developing, even if no individual sensor is reporting an out-of-tolerance situation? The integrator function could report this situation (and even schedule preventative maintenance), avoiding unexpected downtime under future peak loads.

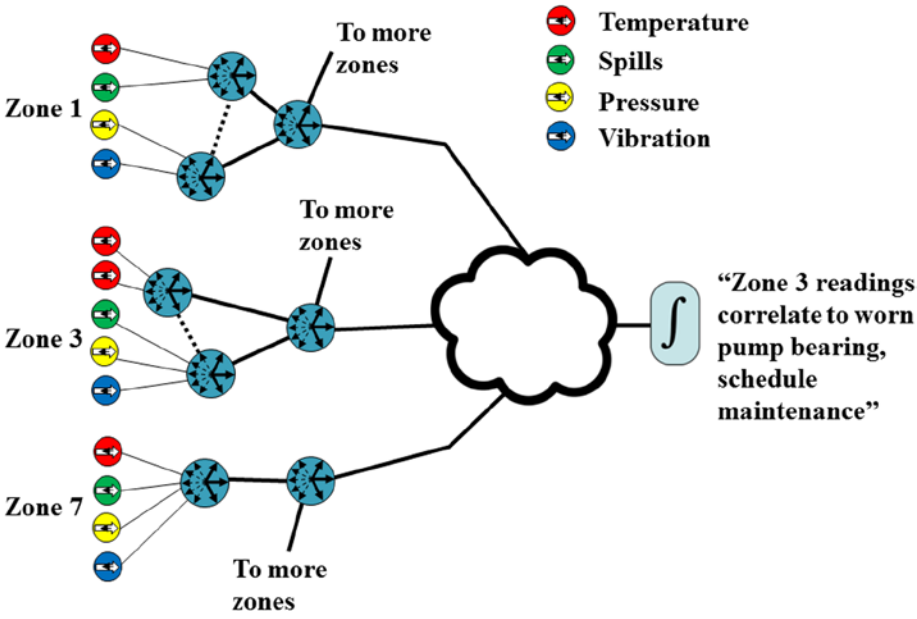

Figure 5-9. *By monitoring thousands of end devices and sensors of various types, an integrator function might infer an impending need for maintenance in a location even though no individual sensor is yet out-of-threshold*

In a complementary way, desired end point settings and configurations may be entered into the integrator function for dissemination across the network. In this case, the integrator function may be given broad commands ("reduce discretionary electrical use"), which results in a wide variety of different end devices at many locations being targets, perhaps in a specific order. In this example, the integrator function might use time-of-day, weather, and other information to determine where and by how much the usage may be cut, rolling the reconfigurations and shutdowns across the globe with regard to the impact of sunlight.

As the name implies, a key characteristic of the integrator function is the intelligent digestion and consolidation of information. For humans interacting with the Internet of Things, the contrasting information of underlying trends and emergent events and alarms are the most important outputs of the integrator function, harnessing the power of myriad IoT end devices.

Machines and Metcalfe

But beyond human interfaces, the IoT integrator function may play a powerful role in pure machine-to-machine networks. In accordance with Metcalfe's Law, the "value" of a communicating network increases with the square of the number of participant members. With large numbers of integrator functions communicating and coordinating with one another, information, resources, and schedules may be shared and optimized without human intervention.

As an example, neighborhood electrical generators plugged into a smart grid energy network with solar panels or wind power sources might be used to support excessive loads during peak times for home appliances. The integrator functions interacting with the various electrical generators and appliances may exchange information to conserve the joint resources and exploit the cheapest sources by studying the patterns in terms of when devices are in use and how much power is typically drawn. The distributed system can thus "schedule" operation into optimal timeslots using Bayesian reasoning.

Over time, machine learning agents within integrator functions may suggest that some competencies be "fused." Fused competencies are, as the name suggests, tightly coupled, largely self-sufficient capabilities between neighborhoods of end devices monitored and controlled by interacting integrator functions.

"Socially networked" integrator functions will also obviously have much broader potential views of events and trends, making possible more useful analysis than any single integrator function.

Collaborative Scheduling Tools

One potentially compelling area for the use of machine-to-machine integrator function interactions is in the area of collaborative scheduling. The example described previously is one instantiation, but broader scheduling efficiencies can be imagined across much broader domains.

The underlying fundamental scheduling principle to be exploited is "cluster" versus "avoid": that is, what activities, events, or elements create more efficiencies when brought together (multiple packages for adjacent addresses sharing the same delivery van, for example) versus those that create more efficiencies when separated (many delivery trucks that must share the same loading dock, for example). By considering a variety of data sources and providing "back pressure" to reschedule or reorder some events or tasks, interacting integrator nodes might allow better use of scarce resources with learning and improved optimization taking place over time.

Packaging and Provisioning

As noted in the introduction, the Internet of Things integrator function is software running on a general-purpose processor with the appropriate performance characteristics and interfaces. With the promulgation of minimal necessary standards and open source code, a wide variety of different organizations and individuals could begin to rapidly create integrator function software to run on many different platforms.

These applications will be programming-intensive to tailor to specific needs, but making available open-source software modules delivering basic functionality will speed deployment. These open-source components are an important part of the Internet of Things development blueprint (see Chapter 8). The possibility of running integrator function software on virtually any device from a smartphone on up permits the analysis and control functionality to scale to any size with off-the-shelf hardware.

Distributed Integrator Functions

To this point, the discussion of integrator functions has assumed a processor location likely some distance (physically and or logically) from the end devices with which it is interacting. And for a significant portion of the Internet of Things, this will likely make sense. As noted elsewhere, for the typical case, data rates will be low, the delivery of any single individual chirp uncritical, and synchronization unimportant. But this will not true everywhere.

Video surveillance and monitoring is one application in which the blasé passenger's description of a bumpy flight is apt: "hours of tedium punctuated by moments of terror." The vast majority of many video surveillance streams are unchanging: the view of a hallway or an unopened gate, perhaps. But the amount of streaming data created by that unchanging scene is substantial, depending on the video CODEC in use.

If all that video data were to be propagated through the Internet to a distant integrator function, the bandwidth, delay, and jitter (variation in delay) would be substantial. But if instead a distributed integrator function were placed at the location of the video camera, substantial processing could be done locally, with only exceptions or events (a human crossing the field of view, for example) generating a message to a distant site and triggering real-time video streaming or recording.

Similarly, process control and other real-time functions might best be served by a localized integrator function that could interpret local conditions from chirps produced by temperature and flow sensors, and then generate chirps to adjust a valve accordingly, as seen in Figure 5-10.

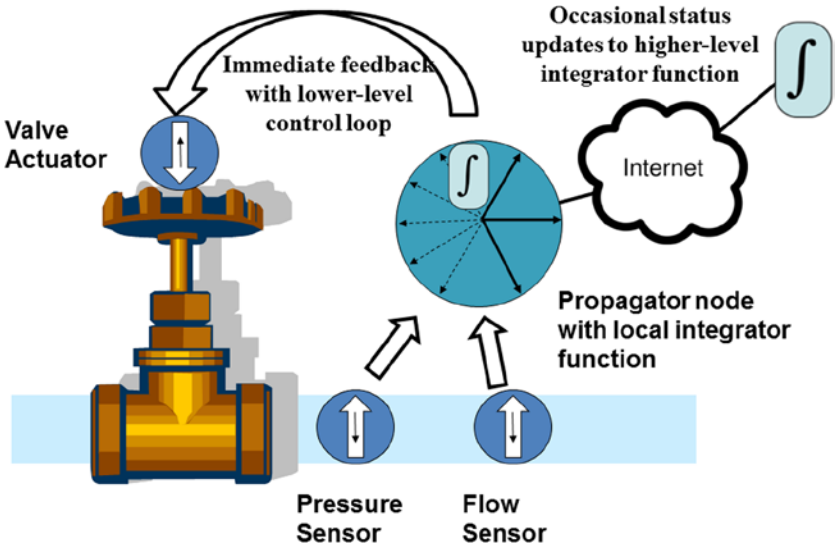

Figure 5-10. *To maximize the response to changing conditions, local flow analysis might take place at an integrator function co-deployed with the nearest propagator. Nominal variations in flow or pressure could be managed by the local action of adjusting valves, whereas periodic status reports and exceptions beyond specific tolerances would be forwarded to an integrator function with a "broader" view*

At the very edges of the Internet of Things, the need for compact integrator function implementations that use a minimum of power and space will demand very small-footprint System-on-a-Chip (SoC) solutions such as Intel's Quark family. These compact microprocessor systems still run standard operating system software and will thus be good targets for rapid development and deployment of distributed integrator function designs, as opposed to fully custom hardware.

In addition to the typical general-purpose processor used for most integrator functions, some distributed integrator functions will certainly also be implemented on customized hardware, often packaged in combination with end device or propagator node hardware.

Location, Location, Location

Application designers determining the optimal location for the IoT integrator function will wish to balance the efficiency of a position near the monitored or controlled devices with the broader perspective that can be gained from placement farther (logically) from the end device. The ability to build a publish/subscribe neighborhood that incorporates varied data sources may outweigh the nominal efficiency of being near the point of analysis or control.

A related decision point is setting the threshold for "phoning home" to a headquarters or oversight location versus managing as best as possible with the information available locally. The incredible diversity of Internet of Things applications will likely create a commensurate variety of deployment approaches.

Filtering the Streams

To make software development and application proliferation easy, the integrator function is specifically architected to operate on general-purpose hardware. Although this type of equipment is well-suited to crunching the large amounts of data potentially generated by thousands or millions of IoT devices, it is generally not optimized for interface to the Internet. Millions of data streams, many of which are of no interest or even ill-intentioned, may arrive at an exposed PC or server Ethernet interface.

In busy applications, handling all this traffic to search out the meaningful IoT small data streams would slow the main processor and reduce its capacity for the main integrator function tasks. So the emerging Internet of Things architecture allows for an additional appliance called the *filter gateway*.

The filter gateway sits between the global Internet and the general-purpose processor (see Figure 5-11). Essentially its function is as a "two-armed" router (for example, Gigabit Ethernet in/Gigabit Ethernet out), providing network service, security, and firewall capabilities. The filter gateway simply discards non-relevant data to reduce the load on the general-purpose hardware running the integrator function software.

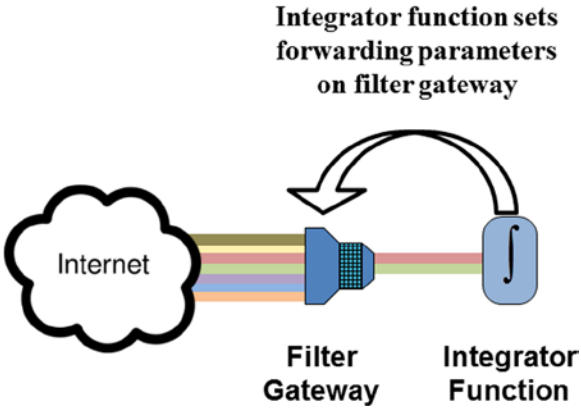

Figure 5-11. *Filter gateways act as firewalls to off-load network interface tasks from the general-purpose processors running integrator function software*

It is likely that existing router and/or security appliance hardware may be adapted to this role. A key software addition to off-the-shelf or open-source devices will be a publishing agent within the filter gateway. It will perform the same function as the publishing agent found in some classes of propagator nodes, allowing the integrator function to "tune" the data streams it sends and receives through the biasing techniques described previously.

Accessing the Power of the Internet of Things

The integrator function turns the numberless streams of data from Internet of Things end devices into rich publish/subscribe information sources and extracts meaning from potential chaos. In the next chapter, the protocols of the emerging Internet of Things architecture are explored in detail.