NIST Special Publication 800-183

Networks of 'Things'

Jeffrey Voas

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-183

C O M P U T E R S E C U R I T Y

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Special Publication 800-183

Networks of 'Things'

Jeffrey Voas
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-183

July 2016

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

Authority

This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C. § 3541 *et seq.*, Public Law (P.L.) 113-283. NIST is responsible for developing information security standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

National Institute of Standards and Technology Special Publication 800-183
Natl. Inst. Stand. Technol. Spec. Publ. 800-183, 30 pages (July 2016)
CODEN: NSPUE2

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-183

Comments on this publication may be submitted to:

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: iot@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

Abstract

System primitives allow formalisms, reasoning, simulations, and reliability and security risk-tradeoffs to be formulated and argued. In this work, five core primitives belonging to most distributed systems are presented. These primitives apply well to systems with large amounts of data, scalability concerns, heterogeneity concerns, temporal concerns, and elements of unknown pedigree with possible nefarious intent. These primitives are the basic building blocks for a Network of 'Things' (NoT), including the Internet of Things (IoT). This document offers an underlying and foundational understanding of IoT based on the realization that IoT involves sensing, computing, communication, and actuation. The material presented here is generic to all distributed systems that employ IoT technologies (i.e., 'things' and networks). The expected audience is computer scientists, IT managers, networking specialists, and networking and cloud computing software engineers. To our knowledge, the ideas and the manner in which IoT is presented here is unique.

Keywords

Internet of Things (IoT); Network of Things (NoT); reliability; security; trust; trustworthiness, sensors; big data; composability; distributed system.

Acknowledgment

Appreciation is given to Matt Scholl, Dave Ferraiolo, Tim Grance, Irena Bojanova, Angelos Stavrou, Rick Kuhn, and Phil Laplante, for the encouragement and support to take on this task. Appreciation is also given to Constantinos Kolia (GMU) for the suggestions in Section 5, and to Svetlana Lukoyanova for the artwork.

Table of Contents

List of Figures

List of Tables

1 Introduction

From agriculture, to manufacturing, to smart homes, to healthcare, and beyond, there is value in having numerous sensory devices connected to larger infrastructures. This technology advance acknowledges the reality that human society is moving towards 'smart' and 'smarter' systems. The rapid advances in computer science, software engineering, systems engineering, networking, sensing, communication, and artificial intelligence are converging. The tethering factor is data.

There is no formal, analytic, or even descriptive set of the building blocks that govern the operation, trustworthiness, and lifecycle of IoT. A composability model and vocabulary that defines principles common to most, if not all networks of things, is needed to address the question: "what is the science, if any, underlying IoT?" This document offers an underlying and foundational science to IoT based on a belief that IoT involves *sensing*, *computing*, *communication*, and *actuation.*

This document uses two acronyms, IoT and NoT (Network of Things), extensively and interchangeably—the relationship between NoT and IoT is subtle. IoT is an instantiation of a NoT, more specifically, IoT has its 'things' tethered to the Internet. A different type of NoT could be a Local Area Network (LAN), with none of its 'things' connected to the Internet. Social media networks, sensor networks, and the Industrial Internet are all variants of NoTs. This differentiation in terminology provides ease in separating out use cases from varying vertical and quality domains (e.g., transportation, medical, financial, agricultural, safety-critical, security-critical, performance-critical, high assurance, to name a few). That is useful since there is no singular IoT, and it is meaningless to speak of comparing one IoT to another.

Primitives are building blocks that offer the possibility of an answer to the aforementioned questions and statements by allowing comparisons between NoTs. We use the term primitive to represent smaller pieces from which larger blocks or systems can be built. For example, in software coding, primitives typically include the arithmetic and logical operations (plus, minus, and, or, etc.). In this document, we do not employ the restriction that primitives cannot be developed or derived from something else; this is often a common, paraphrased definition for "primitive," but it is not employed here.

This model does not specify a definition for what is or is not a 'thing.' Instead, we consider that each primitive injects a behavior representing that 'thing' into a NoT's workflow and dataflow. 'Things' can occur in physical space or virtual space. In physical space, consider humans, vehicles, residences, computer, switches, routers, smart devices, road networks, office buildings, etc. In virtual space, consider software, social media threads, files, data streams, virtual machines, virtual networks, etc. More ideas concerning 'things' is presented in Appendix C.

Primitives offer a unifying vocabulary that allows for composition and information exchange among differently purposed networks. They offer clarity regarding concerns that are subtle, including interoperability, composability, and continuously-binding assets that come and go on-the-fly. Because no simple, actionable, and universally-accepted definition for IoT exists, the model and vocabulary proposed here reveals underlying foundations of the IoT, i.e., they expose the ingredients that can express how the IoT *behaves*, without defining IoT. This offers insights into issues specific to trust.

Further, we employ a paraphrased, general definition for a *distributed system*: a software system in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.[1] NoTs satisfy this definition. Thus we consider IoT to be one type of a NoT and a NoT to be one type of a distributed system.

2 The Primitives

The *primitives* of a NoT are: 1) **Sensor**, 2) **Aggregator**, 3) **Communication channel**, 4) external utility (***e*Utility)**, and 5) **Decision trigger**. There may be some NoTs that do not contain all of these, but that will be rare.

Each primitive, along with its definition, assumptions, properties, and role, is presented. We employ a data-flow model, captured as a sequence of four figures in this document, to illustrate how primitives, when composed in a certain manner, could impact a confidence in trustworthiness of NoTs. Although this model may seem overly abstract at first glance, its simplicity offers a certain elegance by not over complicating IoT's small handful of building blocks.

2.1 Primitive #1: Sensor

A *sensor* is an electronic utility that measures physical properties such as temperature, acceleration, weight, sound, location, presence, identity, etc. All sensors employ mechanical, electrical, chemical, optical, or other effects at an interface to a controlled process or open environment. Basic properties, assumptions, recommendations, and general statements about sensor include:

1. Sensors are physical; some may have an Internet access capability.
2. Sensor output is data; in our writings, $s_1 \rightarrow d_1$ means that sensor 1 has produced a piece of data that is numbered 1. Likewise, $s_2 \rightarrow d_2$ means that sensor 2 has produced a piece of data that is numbered 2. (d_1, d_2, d_3, … d_n will likely be digital data.) Analog sensors such as microphones and voltmeters are counterexamples.
3. A sensor may also transmit device identification information, such as via Radio Frequency Identification (RFID)[2].

[1] George Coulouris et al., *Distributed Systems: Concepts and Design*, 5th ed. (Boston: Addison-Wesley, 2011), 2.

[2] RFID is an automatic identification method that stores and remotely retrieves data via a RFID tag or transponder. A RFID programmer encodes information onto a tiny microchip within a thin RFID inlay. In supply chain applications, these inlays typically are embedded in a tag that looks similar to pressure sensitive labels. RFID inlays can be applied to a wide variety of NoTs. RFID technology offers security and reliability features that enhance trustworthiness in IoT ecosystems.

4. Sensors may have an identity or have the identity of the 'thing' to which they are attached.
5. Sensors may have little or no software functionality and computing power; more advanced sensors may have software functionality and computing power.
6. Sensors may be heterogeneous, from different manufacturers, and collect data, with varying levels of data integrity.
7. Sensors may be associated with fixed geographic locations or may be mobile.
8. Sensors may provide surveillance. Cameras and microphones are sensors.
9. Sensors may have an owner(s) who will have control of the data their sensors collect, who is allowed to access it, and when.
10. Sensors will have pedigree – geographic locations of origin and manufacturers. Pedigree may be unknown, and suspect.
11. Sensors may be cheap, disposable, and susceptible to wear-out over time.
12. There may be differentials in sensor security, safety, and reliability, e.g., between consumer grade, military grade, industrial grade, etc.
13. Sensors may return no data, totally flawed data, partially flawed data, or correct and acceptable data. Sensors may fail completely or intermittently. They may lose sensitivity or calibration.
14. Sensors are expected to return data in certain ranges, e.g., [1 … 100]. When ranges are violated, rules may be needed on whether to turn control over to a human or machine when ignoring out-of-bounds data is inappropriate.
15. Sensors may be disposable or serviceable in terms of calibration, sensitivity or other forms of refresh. Complex and expensive sensors may be repaired instead of replaced.
16. Sensors may be powered in a variety of ways including alternating current (AC), solar, wind, battery, or passively via radio waves.
17. Sensors may be acquired off-the-shelf or built to specification.
18. Sensors acquire data that can be event-driven, driven by manual input, command-driven, or released at pre-defined times.
19. Sensors may have a level of data integrity ascribed (see Section 2.2.2).
20. Sensors may have their data encrypted to void some security concerns.

21. Sensors should have the capability to be authenticated as genuine.

22. Sensor data may be sent and communicated to multiple NoTs. A sensor may have multiple recipients of its data. Sensor data may be leased to one or more NoTs.

23. The frequency with which sensors release data impacts the data's currency and relevance. Sensors may return valid but stale data. Sensor data may be 'at rest' for long periods of time.

24. A sensor's precision may determine how much information is provided. *Uncertainty* of sensor data should be considered.

25. Sensors may transmit data about the "health" of a system, such as is done in prognostics and health management (PHM).

26. In this document and model, we do not classify humans as sensors; humans are classified as an *e*Utility, our fourth primitive (see Section 2.4), and/or classified as our fourth element, owner (see Section 3). When classified as an *e*Utility, humans can still act in a sensor-like role by manually feeding data into a NoT's workflow and dataflow.

27. Humans can influence sensor performance through failure to follow policy, sensor misplacement, etc. (or their positive analogs). Humans are potential contributors to sensor failures.

28. Security is a concern for sensors if they or their data is tampered with, stolen, deleted, dropped, or transmitted insecurely so it can be accessed by unauthorized parties. Building security into specific sensors may or may not be necessary based on the overall system design.

29. Reliability is a concern for sensors.

2.2 Primitive #2: Aggregator

An *aggregator* is a software implementation based on mathematical function(s) that transforms groups of *raw* data into *intermediate, aggregated* data. Raw data can come from any source. Aggregators help in managing 'big' data. Basic properties, assumptions, recommendations, and general statements about aggregator include:

1. Aggregators may be virtual due the benefit of changing implementations quickly and increased malleability. A situation may exist where aggregators are physically manufactured, e.g., a field-programmable gate array (FPGA) or hard-coded aggregator that is not programmable. Aggregators may also act in a similar way as *n*-version voters.

2. Aggregators require computing horsepower, however this assumption can be relaxed by changing the definition and assumption of virtual to physical, e.g. firmware,

microcontroller or microprocessor. For example, aggregators could execute on faster hardware such as a smartphone. Aggregators will likely use weights (see Section 2.2.2) to compute intermediate, aggregated data.

3. Aggregators have two actors for consolidating large volumes of data into lesser amounts: Clusters (see Section 2.2.1), and Weights (see Section 2.2.2). Aggregators process *big data* concerns within NoTs, and to satisfy this role computational "performance enhancing" technologies will be needed. This is the only primitive with actors.

4. Sensors may communicate directly with other sensors, and thus act in some situations quite similar to aggregators.

5. Intermediate, aggregated data may suffer from some level of *information loss*. Proper care in the aggregation process should be given to significant digits, rounding, averaging, and other arithmetic operations to avoid unnecessary loss of precision.

6. For each cluster (see Section 2.2.1) there should be an aggregator or set of potential aggregators.

7. Aggregators are: (1) executed at a specific time and for a fixed time interval, or (2) event-driven.

8. Aggregators may be acquired off-the-shelf. Note that aggregators may be non-existent and will need to be home-grown. This may create a problem for huge volumes of data within a NoT.

9. Some NoTs may not have an aggregator, e.g., a single light sensor will send a signal directly to a smart lightbulb to turn it off or on.

10. Security is a concern for aggregators (malware or general defects) and for the sensitivity of their aggregated data. Further, aggregators could be attacked, e.g., by denying them the ability to operate/execute or by feeding them bogus data.

11. Reliability is a concern for aggregators (general defects).

2.2.1 Actor #1: Cluster (or "Sensor Cluster")

A *cluster* is an abstract grouping of sensors that can appear and disappear instantaneously. Basic properties, assumptions, recommendations, and general statements about cluster include:

1. Clusters are abstractions of a set of sensors along with the data they output—clusters may be created in an *ad hoc* manner or organized according to fixed rules.

2. Clusters are not inherently physical.

3. C_i is essentially a *cluster* of the sensor data from $n \geq 1$ sensors, $\{d_1, d_2, d_3, \ldots, d_n\}$.

4. C_i may share one or more sensors with C_k, where $i \neq k$, or with other NoTs. This is somewhat important, because competing organizations may be receiving data that they believe to be unique and purposed only for them to receive, and not realizing a competitor is also receiving the same sensor data.

5. *Continuous-binding* of a sensor to a cluster may result in little ability to mitigate trustworthiness concerns of a real-time NoT if the binding occurs *late*.

6. Clusters are malleable and can change their collection of sensors and their data at any time.

7. The composition of clusters is dependent on what mechanism is employed to aggregate the data, which ultimately impacts the purpose and requirements of a specific NoT.

Note item 4 in the above list; it is subtly important – it relates to business competition for highly valued data.

2.2.2 Actor #2: Weight

Weight is the degree to which a particular sensor's data will impact an aggregator's computation. Basic properties, assumptions, recommendations, and general statements about weight include:

1. A weight may be hardwired or modified on-the-fly.

2. A weight may be based on a sensor's perceived trustworthiness, e.g., based on who is the sensor's owner, manufacturer, geographic location where the sensor is operating, sensor age or version, previous failures or partial failures of sensor, sensor tampering, sensor delays in returning data, etc. A weight may also be based on the worth of the data, uniqueness, relation to mission goals, etc.

3. Different NoTs may leverage the same sensor data and re-calibrate the weights per the purpose of a specific NoT.

4. It is not implied that an aggregator is necessarily a functionally linear combination of sensor outputs. Weights could be based on other logical insights, such as the following: if sensor A output is greater than 1 use Sensor B output else use Sensor C's output.

5. Aggregators may employ artificial intelligence techniques to modify their clusters and weights on-the-fly.

6. Weights will affect the degree of information loss during the creation of intermediate data.

7. Redundant sensors may increase a sensor's weight if a grouping of redundant sensor data is in agreement and produces the same result. Repeated sampling of the same sensor might also affect a sensor's weighting, either positively or negatively, depending on the continuity of a particular output value during a fixed time interval.

8. Security concerns for weights is related to possible tampering of the weights.

9. The appropriateness (or correctness) of the weights is crucial for the purpose of a NoT.

A simple aggregator might implement the summation

$$\sum_{i=1}^{x} d_i$$

divided by *x*, where *x* is the number of data points, and where the weight for each data point is *uniform*.

2.3 Primitive #3: Communication Channel

A *communication channel* is a medium by which data is transmitted (e.g., physical via Universal Serial Bus (USB), wireless, wired, verbal, etc.). Basic properties, assumptions, recommendations, and general statements about communication channel include:

1. Communication channels move data between computing, sensing, and actuation.

2. Since data is the "blood" of a NoT, communication channels are the "veins" and "arteries", as data moves to and from intermediate events at different snapshots in time. We talk more about the role of snapshots in time in Section 3.

3. Communication channels will have a physical or virtual aspect to them, or both. Protocols and associated implementations provide a virtual dimension. Wires provide a physical dimension.

4. Communication channel dataflow may be unidirectional or bi-directional. There are a number of conditions where an aggregator might query more advanced sensors, or potentially recalibrate them in some way (e.g., request more observations per time interval).

5. No standardized communication channel protocol is assumed; a specific NoT may have multiple communication protocols between different entities.

6. Communication channels may be wireless.

7. Communication channels may be an offering (*service* or *product*) from third-party vendors.

8. Communication channel *trustworthiness* may make sensors appear to be failing when actually the communication channel is failing.

9. Communication channels can experience disturbances, delays, and interruptions.

10. *Redundancy* can improve communication channel reliability. There may be more than one distinct communication channel between a computing primitive and a sensing primitive.

11. Performance and availability of communication channels will greatly impact any NoT that has time-to-decision requirements (see the Decision trigger primitive in Section 2.5).

12. Security and reliability are concerns for communication channels.

In Figure 1, 15 sensors are shown – the blue sensors indicate that these 2 sensors are 'somehow' failing and at specific times, that is, they are not satisfying their purpose and expectations. As mentioned earlier, there could be a variety of sensor failure modes, some temporal, and some related to data quality. Further the temporal failure modes for sensors may be actually a result of the transport of that data failing, and not the sensors themselves. Consider also that the two failing sensors in Figure 1 should probably be assigned lower weights. Figure 1 also shows the 15 sensors clustered into 3 clusters with 5 unique sensors assigned to each. Figure 1 shows the data coming out from each of the three clusters as being inputted to 3 corresponding aggregators. It is now the responsibility of the 3 aggregators to turn those 15 sensor inputs into 3 intermediate data points.

Note the close relationship between clusters and aggregators. For example, in Figure 1, aggregator A_1 might be determining how busy restaurant *X* is. Five independent sensors in C_1 could be taking pictures from inside and outside (parking lot) of *X*, room temperature measurement in the kitchen, motion detectors from the dining area, sound and volume sensors, light detectors, etc. So while the sensors are certainly not homogeneous, their data is processed to make a new piece of data to address one question with possible results such as is the restaurant busy, not busy, closed, etc. And aggregators A_2 and A_3 might be doing the same for restaurants *Y* and *Z* respectively. Consider also that an aggregator can be associated with different clusters.

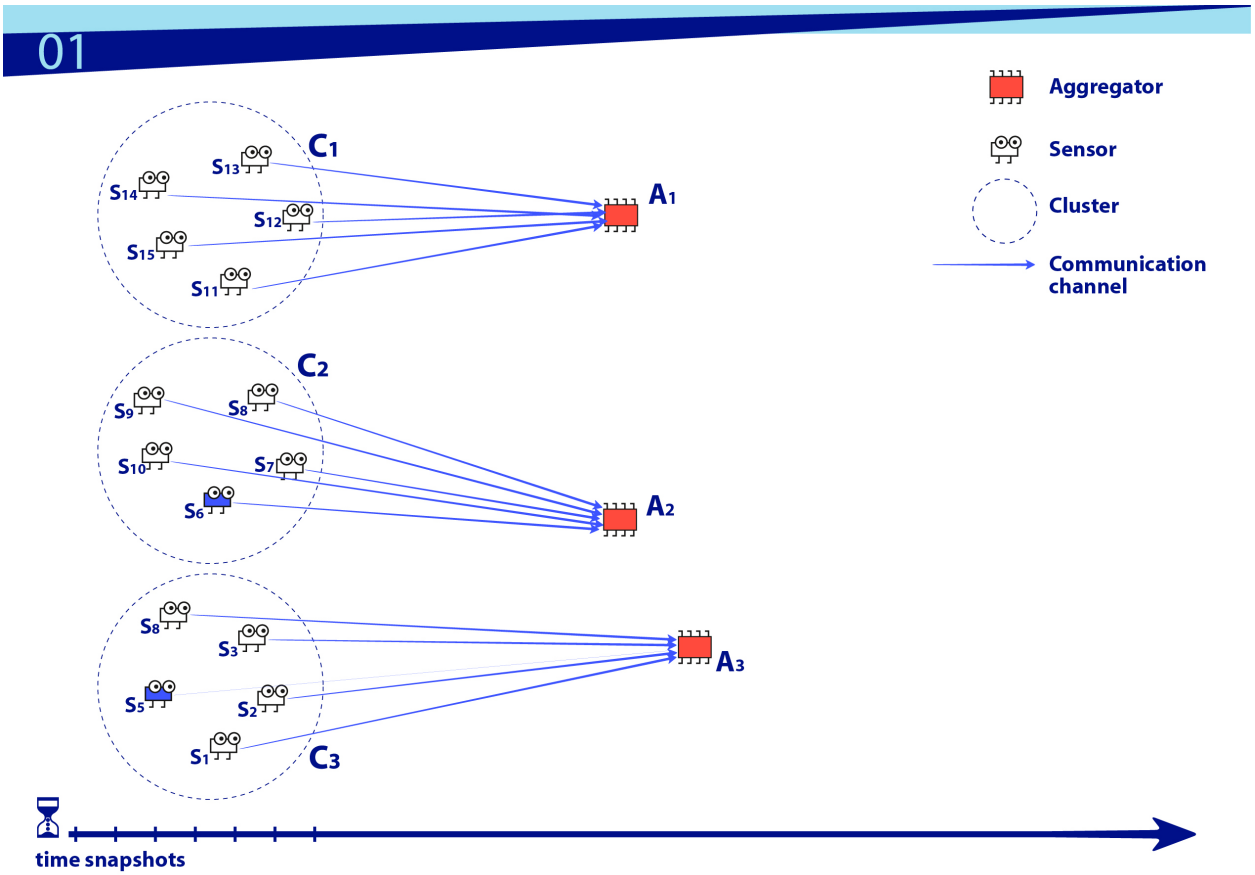

Figure 1: The first three primitives

2.4 Primitive #4: *e*Utility (external utility)

An *e*Utility (external utility*)* is a software or hardware product or service.[3] The current definition of *e*Utility is deliberately broad, to allow for unforeseen future services and products that will be incorporated in future types of NoTs yet to be defined. (Breaking this primitive down into sub-primitives is future work.) Basic properties, assumptions, recommendations, and general statements about *e*Utility include:

1. *e*Utilities execute processes or feed data into the overall workflow of a NoT.

2. *e*Utilities could be acquired off-the-shelf from 3rd parties.

[3] Standards Developing Organizations (SDOs) have referred to *e*Utilities as *cyber-entities* [IEEE p2413, "Standard for an Architectural for the Internet of Things (IoT)] and *digital entities* [JTC21 WG10 ISO/IEC 34010, "Internet of Things"] in their draft documents as of this writing.

3. *e*Utilities may include databases, mobile devices, misc. software or hardware systems, clouds, computers, CPUs, etc. The *e*Utility primitive can be subdivided, and probably should be decomposed to make this model less abstract.

4. *e*Utilities, such as clouds, provide computing power that aggregators may not have.

5. A human may be viewed as a *e*Utility. A human is sometimes considered as a 'thing' in public discourse related to IoT.

6. Data supplied by an *e*Utility may be weighted.

7. An *e*Utility may be counterfeit; this is mentioned later in element Device_ID (see Section 3).

8. Non-human *e*Utilities may have Device_IDs; Device_IDs may be crucial for identification and authentication.

9. Security and reliability are concerns for all *e*Utilities.

Figure 2 illustrates using two cloud *e*Utilities to execute five aggregator implementations. (Note that there is no suggestion here that aggregators must execute on cloud platforms – this was for illustration only – aggregators can execute on any computing platform.) Figure 2 shows the addition of one non-cloud *e*Utility, eU_1 (a laptop).

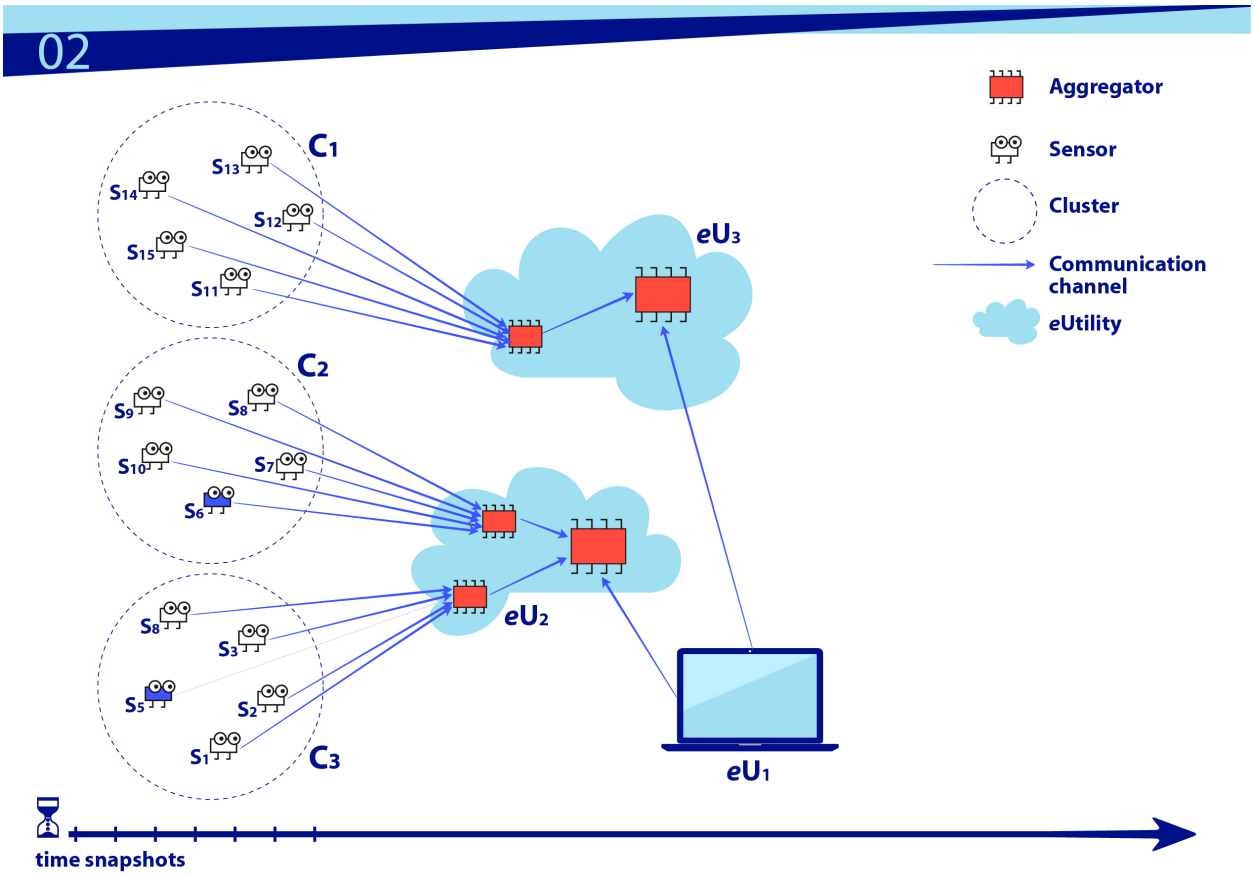

Figure 2: *e*Utility

2.5 Primitive #5: Decision Trigger

A *decision* trigger creates the final result(s) needed to satisfy the purpose, specification, and requirements of a specific NoT. Basic properties, assumptions, recommendations, and general statements about decision trigger include:

1. A decision trigger is a conditional expression that triggers an action. A decision trigger's outputs can control actuators and transactions (see Figure 3 and Figure 4). Decision triggers abstractly define the end-purpose of a NoT.

2. A NoT, may or may not control an actuator (via a decision trigger).

3. A decision may have a binary output, but there will also be situations where the output of a decision trigger is a non-discrete output, i.e., a continuum of output values.

4. A decision trigger may have a built-in adaption capability as the environment element (see Section 3) changes.

5. A decision trigger will likely have a corresponding virtual implementation, i.e., code.

6. A decision trigger may have a unique owner.
7. Decision triggers may be acquired off-the-shelf or homegrown.
8. Decision triggers are executed at specific times or may execute continuously as new data becomes available.
9. Decision trigger results may be predictions.
10. Analytics could be implemented within decision triggers, however analytics could also be implemented within aggregators (that are executed by *e*Utilities).
11. If a decision trigger feeds data signals into an actuator, then the actuator may be considered as a *e*Utility if the actuator feeds data back into the NoT. Also, an actuator can and often should be considered as a component of the environment element (see Section 3). This model treats actuators as "consumers" of the outputs from decision triggers. Actuators are 'things,' but not all things are primitives in this model.
12. A decision trigger may feed its output back into the NoT creating a feedback loop[4] (See Figure 4).
13. It is fair to view a decision trigger as an *if-then* rule, although they will not all have this form.
14. The workflow up to decision trigger execution may be partially parallelizable.
15. Failure to execute decision triggers at time t_x may occur due to tardy data collection, inhibited sensors or *e*Utilities, inhibited communication channels, low performance aggregators, and a variety of other subsystem failure modes.
16. Economics and costs can play a role in the quality of the decision trigger's output.
17. There may be intermediate decision triggers at any point in a NoT's workflow.
18. Decision triggers act similarly to aggregators, and can be viewed as a special case of aggregator.
19. Security is a concern for decision triggers (malware or general defects). Other possibilities here might be indirect manipulation of input values to the trigger by tampering with or restricting the input values.

[4] There are other *feedforward* mechanisms in process control theory and implementation that are outside the immediate scope of this model.

20. Reliability is a concern for decision triggers (general defects). Decision triggers could be inconsistent, self-contradictory, and incomplete. Understanding how bad data propagates to affect decision triggers is paramount. Failure to execute decision triggers at time t_x may have undesired consequences.

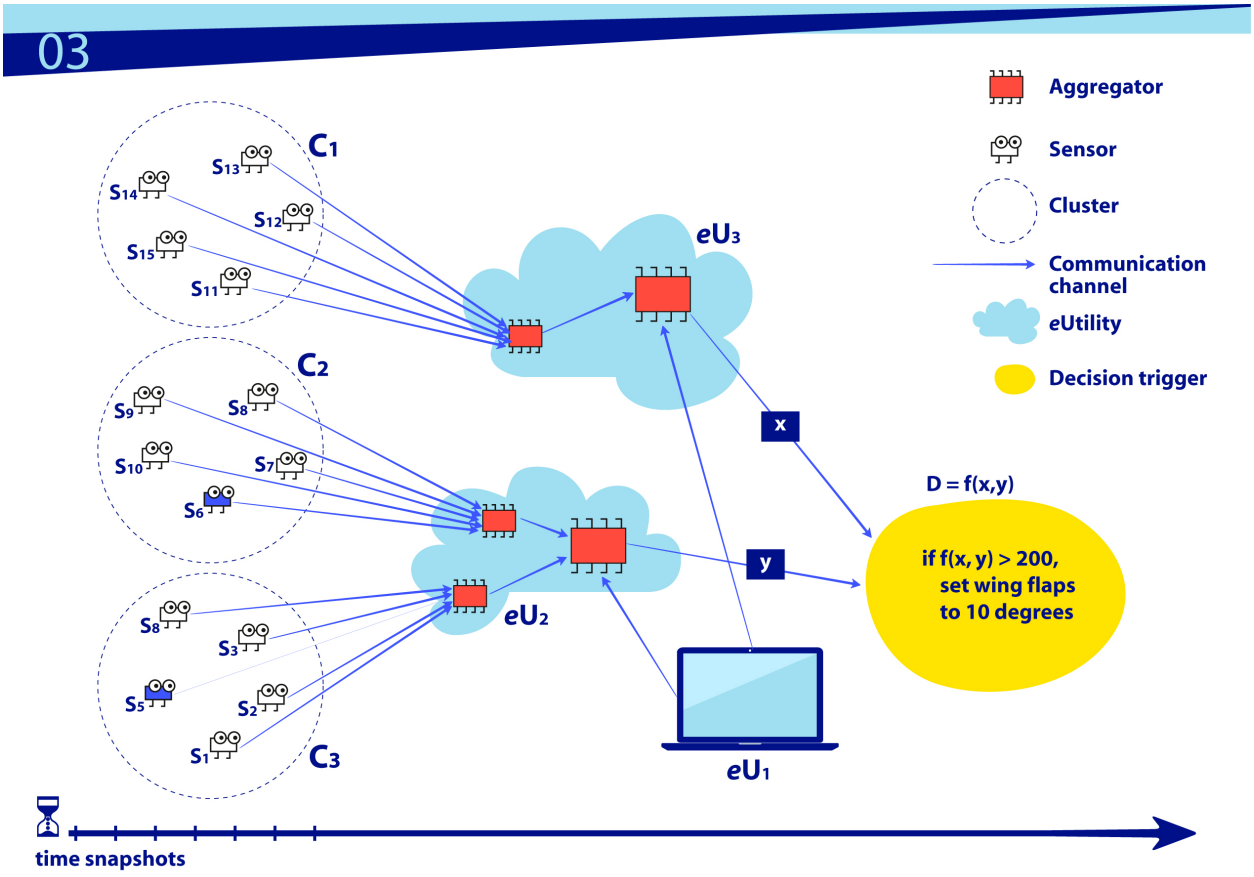

Figure 3: Decision trigger

Decision triggers are predicates that must be *true* to initiate a command or action. They will frequently be represented as *if-then* rules (although this may not always be the case), and variables within the predicate may represent data or sensor values from parts of the NoT. Reliability is essential, indicating the need for strong testing and assurance of decision triggers. Fortunately, methods and tools for testing access control rules are well developed and may apply to a large set of decision trigger testing problems [Kuhn 2016]. Paralleling the structure of decision triggers, access control rules typically take the form of predicates evaluating to two-valued decisions – either *grant* or *deny* in most cases. For example, a building access control system may include a rule such as, "*if* subject is an employee *and* time is between 6 am and 9 pm *then* grant access." Access control rules are composed into policies, which may include hundreds of rules with a large number of variables interacting in complex ways. Methods and tools exist to check complex access control policies for consistency and completeness, and to generate tests for assurance of policies implemented in code. These access control policy testing tools could be adapted to NoT verification and testing, providing a level of assurance that will be essential for the more critical NoT applications.

Figure 4 shows an alternative to any suggestion that this model of a NoT's dataflow is necessarily uni-directional; it depicts a decision trigger that actually feeds its computation for *g*(x, y) back into the NoT, creating a continuous feedback loop. So for example if new sensor data were fed continuously into a NoT's workflow and dataflow, that data can be combined with the results of previous decision trigger outputs to create updated decision trigger results at later points in time.

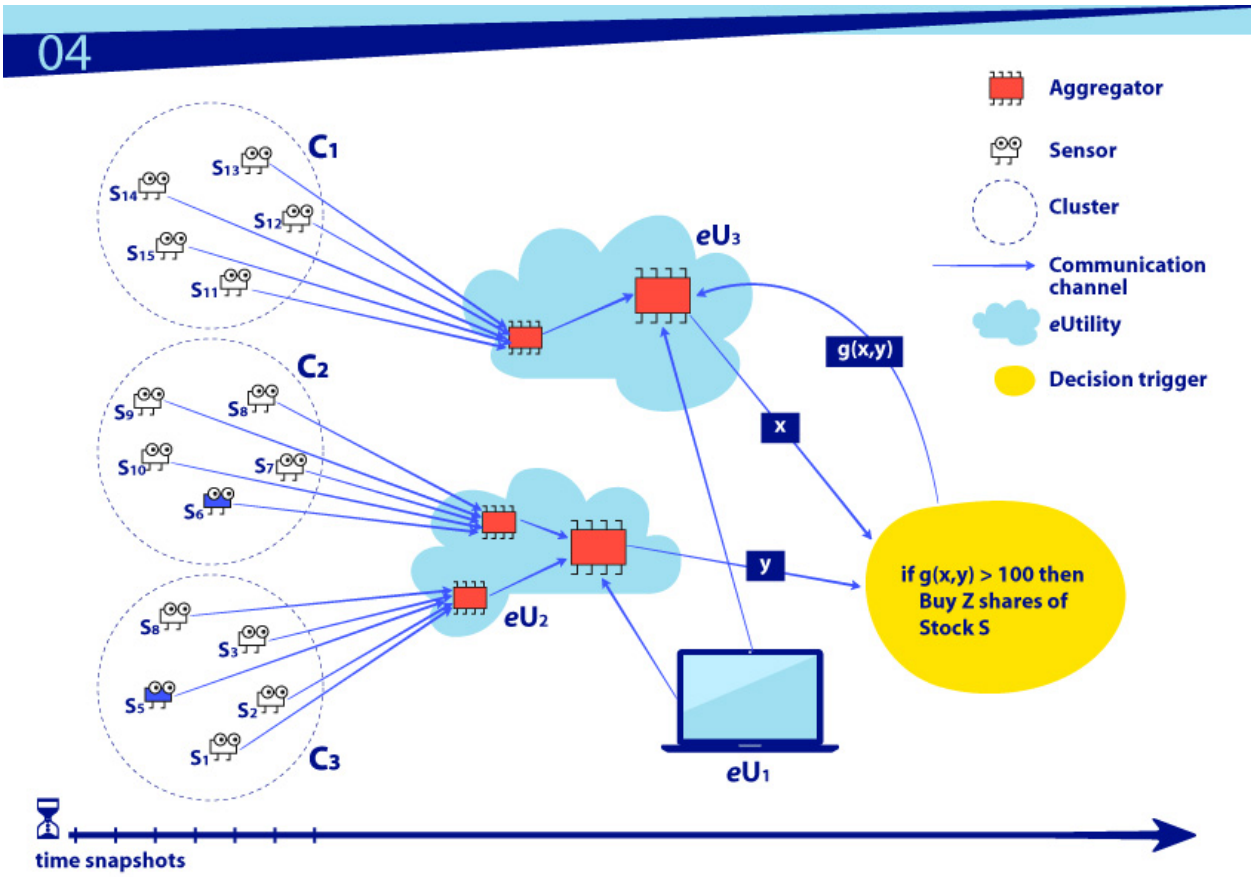

Figure 4: Decision trigger with feedback

2.6 Additional Notes on the Primitives

Now, a few additional points concerning the interplay and relationship between the five primitives are as follows. First, sensor feeds aggregator. Secondly, aggregator executes on various *e*Utilities of a NoT. Thirdly, communication channels are the veins and arteries that connect sensor, aggregator, *e*Utility, and decision trigger with the data that flows between them. And fourth, sensor, aggregator, communication channel, *e*Utility, and decision trigger all have events firing at specific snapshot times; a large challenge for IoT and NoTs is to keep these events in sync, just as for any distributed system.

It is important to treat all events with respect to their temporal location, i.e., their geographical place and at what time. For example, the older events become, the staler or less interesting they

may be relative to immediate decision making. Historical information may be useful in other calculations such as statistical reporting or model generation.

3 The Elements

To complete this model, we define six elements: *environment*, *cost*, *geographic location*, *owner*, *Device_ID*, and *snapshot*, that although are not primitives, are key players in trusting NoTs. These elements play a major role in fostering the degree of trustworthiness[5] that a specific NoT can provide.

1. **Environment** – The universe that all primitives in a specific NoT operate in; this is essentially the *operational profile* of a NoT. The environment is particularly important to the sensor and aggregator primitives since it offers context to them. An analogy is the various weather profiles that an aircraft operates in or a particular factory setting that a NoT operates in. This will likely be difficult to correctly define.

2. **Cost** – The expenses, in terms of time and money, that a specific NoT incurs in terms of the non-mitigated reliability and security risks; additionally, the costs associated with each of the primitive components needed to build and operate a NoT. Cost is an estimation or prediction that can be measured or approximated. Cost drives the design decisions in building a NoT.

3. **Geographic location –** Physical place where a sensor or *e*Utility operates in, e.g., using RFID to decide where a 'thing' actually resides. Note that the operating location may change over time. Note that a sensor's or *e*Utility's geographic location along with communication channel reliability and data security may affect the dataflow throughout a NoT's workflow in a timely manner. Geographic location determinations may sometimes not be possible. If not possible, the data should be suspect.

4. **Owner** - Person or Organization that owns a particular sensor, communication channel, aggregator, decision trigger, or *e*Utility. There can be multiple owners for any of these five. Note that owners may have nefarious intentions that affect overall trust. Note further that owners may remain anonymous. Note that there is also a role for an **operator**; for simplicity, we roll up that role into the owner element.

5. **Device_ID** – A unique identifier for a particular sensor, communication channel, aggregator, decision trigger, or *e*Utility. Further, a Device_ID may be the only sensor data transmitted. This will typically originate from the manufacturer of the entity, but it could be modified or forged. This can be accomplished using RFID[6] for physical primitives.

[5] *Trustworthiness* includes attributes such as security, privacy, reliability, safety, availability, and performance, to name a few.

[6] RFID readers that work on the same protocol as the inlay may be distributed at key points throughout a NoT. Readers activate the tag causing it to broadcast radio waves within bandwidths reserved for RFID usage by individual governments internationally. These radio waves transmit identifiers or codes that reference unique information associated with the item to which the RFID inlay is attached, and in this case, the item would be a primitive.

6. **Snapshot** – an instant in time. Basic properties, assumptions, and general statements about snapshot include:

    a. Because a NoT is a distributed system, different events, data transfers, and computations occur at different snapshots.

    b. Snapshots may be aligned to a clock synchronized within their own network [NIST 2015]. A global clock may be too burdensome for sensor networks that operate in the wild. Others, however, argue in favor of a global clock [Li 2004]. This publication does not endorse either scheme at the time of this writing.

    c. Data, without some "agreed upon" time stamping mechanism, is of limited or reduced value.

    d. NoTs may affect business performance – sensing, communicating, and computing can speed-up or slow-down a NoT's workflow and therefore affect the "perceived" performance of the environment it operates in or controls.

    e. Snapshots maybe tampered with, making it unclear when events actually occurred, not by changing time (which is not possible), but by changing the recorded time at which an event in the workflow is generated, or computation is performed, e.g., sticking in a **`delay()`** function call.

    f. Malicious latency to induce delays, are possible and will affect when decision triggers are able to execute.

    g. Reliability and performance of a NoT may be highly based on (e) and (f).

4 Additional Considerations

Five additional considerations to the previous material that defined the primitives and elements include:

1. **Open, Closed**

    NoTs can be open, closed, or somewhere in between. For example, an automobile can have hundreds of sensors, numerous Central Processing Unit (CPU)s, databases such as maps, wired communication channels throughout the car, and without any wireless access between any 'thing' in the car to the outside. This illustrates a closed NoT. Such a NoT mitigates nearly all wireless security concerns such as remotely controlling a car, however there could still be concerns of malware and counterfeit 'things' that could result in reduced safety. (There are issues related to wireless transmission ranges that we ignore here for simplification.) A fully open system would essentially be any 'thing' interoperating with any 'thing,' any way, and at any time. This, from a "trustworthiness" standpoint, is impossible to assure since the NoT is unbounded.

Most NoTs will be between these extremes since a continuum will likely exist. The primitives serve as a guidepost as to where reliability and security concerns require additional mitigation, e.g., testing and other technologies related to validation and verification.

2. Patterns

We envision a future demand for design patterns that allow larger NoTs to be built from smaller NoTs, similar to design patterns in object-oriented systems. In essence, these smaller entities are sub-NoTs. Sub-NoTs could speed-up IoT adoption for organizations seeking to develop IoT-based systems by having access to sub-NoT catalogues. Further, the topology of sub-NoTs could impact the security and performance of composite NoTs.

3. Composition and *Trust*

To understand the inescapable *trust* issues associated with IoT and a specific NoT, consider the attributes of the primitives and elements shown in Table 1. The three rightmost columns are our best guess as to whether the pedigree, reliability, or security of an element or primitive creates a trustworthiness risk. Those with question marks (?) are ones do not think we fully understand at this time.

The following table poses questions such as: *what does trust mean for a NoT when its primitives are in continual flux due to natural phenomenon that are in continuous change and while its virtual and physical entities are unknown, partially unknown, or faulty? Or if we have insecure physical systems employing faulty snapshots composed with incorrect assumed environments, where is the trust?*

Table 1: Primitive and Element Trust Questions

Primitive or Element	Attribute	Pedigree Risk?	Reliability Risk?	Security Risk?
Sensor	Physical	Y	Y	Y
Aggregator	Virtual	Y	Y	Y
Communication channel	Virtual and/or Physical	Y	Y	Y
***e*Utility**	Virtual or Physical	Y	Y	Y
Decision trigger	Virtual	Y	Y	Y

Geographic location	Physical (possibly unknown)	N/A	Y	Y
Owner	Physical (possibly unknown)	?	N/A	?
Environment	Virtual or Physical (possibly unknown)	N/A	Y	Y
Cost	Partially known	N/A	?	?
Device_ID	Virtual	Y	Y	Y
Snapshot	Natural phenomenon	N/A	Y	?

Such questions demonstrate the difficulty assuring and assessing NoT trustworthiness. We offer the following statement about NoT trustworthiness:

Trust in some NoT *A*, at some snapshot *X*, is a function of NoT *A*'s assets ϵ {sensors (s), aggregator(s), communication channel(s), *e*Utility(s), decision trigger(s)} with respect to the members ϵ {geographic location, owner, environment, cost, Device_IDs, snapshot} when applicable.

4. NoT Testability

A testability[7] metric that applies here is titled the Domain Range Ratio [Voas 1993]. This ratio is simply the cardinality of the set of all possible test cases to the cardinality of the set of all possible outputs.

To understand this metric, consider a NoT's decision trigger that forces an actuator to be in one of two states: ('1') or ('0'). (This situation occurs in Figures 3 and 4.) Further, assume that each output state occurs 50 % of the time. Because of this minimal output space size, a fair coin toss also has a 50-50 chance of providing a correct output for any given input, and that's likely less expensive than building a complex NoT. Worse, consider the scenario where '1' and '0' are not evenly distributed, e.g., the specification states that for 1 million unique test cases only 10 should produce a '1' and the other 999 990 should produce a '0'. Here, you could build a NoT to compute this function or you could write a piece of code that simply says: **`for all inputs output ('0')`**. This incorrect software implementation is still 99.999 % reliable, and you'll almost certainly not discover the code defect with a handful of random tests sampled from the 1M. In short, testing here has a minimal probability of detecting the faulty logic because

[7] *Testability* here refers to the likelihood that defects can be discovered during testing [Voas 1995].

each test case has low "detectability" due to the tiny output space and the probability density function for each output.

This shows that because of the decision trigger primitive, specifically purposed NoTs may be inherently untestable without well-placed internal self-tests (assertions[8]) upon the other 4 primitives to increase testability.

5. Environment

As NoTs are likely to rely on factors from outside native environmental boundaries (e.g., sensor readings from a NoT's surrounding external environment, whatever that environment may be), it is equally logical to assume that some form of actuation may be in place to effect or at least influence that external environment as a result of a NoT action. Thus, it is feasible that an *e*Utility may be more than a software-driven virtual entity as is strongly implied, but could also a cyber-physical or purely a mechanical actuator (e.g. control surface hydraulics on an aircraft) that extends an action to influence a NoT environment as opposed to only influencing the internal workings of the NoT itself. This raises the larger question of how a NoT and its environment interact, not only at the element level which is addressed, but, more significantly, at the NoT's operating level.

5 Reliability and Security Primitive Scenarios

The elements lay out key contextual issues related to trustworthiness of a specific NoT. And the primitives are the building blocks of NoTs. Because trustworthiness is such a broad concept, this document has mainly focused on two "ilities" related to the five primitives: security and reliability. People often ask for simple examples of real or hypothetical use cases relating these two "ilities" to each primitive. The following are examples of simple, hypothetical reliability and security scenarios associated with each primitive

Sensor reliability*:* A modern car's speed sensor is exposed to heat, water, and dust (environment). Years later, it starts providing inconsistent readings due to naturally occurring fatigue that induces corrupt sensor data. This is an example of malfunctions caused by environmental conditions.

Sensor security*:* A smart building's temperature sensors are easily accessible and this particular system doesn't provide a means for validating the firmware's authenticity. An attacker substitutes the firmware with one that responds to remote commands. These sensors then become part of a botnet and can contribute to distributed denial-of-service (DDoS) attacks. This is an example of physical tampering and altering firmware.

Aggregator reliability*:* In a smart city environment, thousands of sensors transmit data to a series of smart gateways that effectively compress several gigabytes of raw data into meaningful information. A blackout that occurred in part of the city creates an unexpected condition that results in division by zero, which causes the application to keep crashing for the entire duration of the

[8] Assertions act as "mini-oracles," because they test internal states during execution and add to the argument that the final output was indeed legitimately generated since the internal states that lead to that output were also correct and not dirty.

blackout. This is an example of unpredicted conditions that lead to undefined behavior and incorrect output.

***Aggregator security**:* An attacker introduces a rogue sensor into a network that produces fake readings. These readings are passed as inputs to the aggregator function without any validation. The attacker launches a buffer overflow attack to gain root access to the entire middleware infrastructure (gateway). This is an example of an injection attack or buffer overflow.

***Communication channel reliability**:* 'Smart building' sensors for regulating lights and temperature communicate wirelessly via IEEE 802.11 with the rest of the building management system. During a conference, a large number of people are gathered inside a room, having enabled Wi-Fi on their smartphones. Due to overpopulation of the channel, there are frequent disconnections and service degradation. As a result, the sensors are unable to provide readings with their predefined frequency. This is an example of loss of service due to overpopulation and connection problems.

***Communication channel security**:* A wearable activity tracker is attached to a person's wrist and measures heart rate and blood pressure. It communicates via Bluetooth Low Energy (BLE) with the wearer's smartphone and forwards the data to a physician. Despite the fact that BLE takes specific actions to randomize the MAC address of the devices, the manufacturer neglected this feature. An attacker with a high-gain antenna can track the presence of the wearer in a crowd and create a movement profile. This is an example of eavesdropping on the communication channel.

***eUtility reliability**:* A point-of-sale system conducting automatic smart payments depends on a cloud service for verifying the identity of the person using a card. System maintenance of the cloud server happens to occur during business hours, which causes delays in verification. This is an example of system failures that make the resource unavailable, and therefore service unreliable.

***eUtility security**:* A 'smart home' has a security camera installed at the front door that sends data to a corresponding cloud application that then forwards notifications and video footage to the homeowner's device after motion at the door is detected. An attacker conducts a DDoS attacks on the application provider's servers for two hours. They're able to break into the house without the user being notified. This is an example of a DDoS attack.

***Decision trigger reliability**:* The logic implemented, for example, a decision trigger is just a conditional expression, e.g., **`if a > 100 open garage door`**. But what if the expression should have been **`if a > 10 open garage door`**. This is a reliability problem because the function implemented is incorrect for all values of **`a`** between 11 and 99. Decision triggers are likely to be written in code, although this assumption could be relaxed.

***Decision trigger security**:* The decision trigger implementation accepts malicious inputs or potentially the outputs from the trigger are sniffed and released to competitors unbeknownst to the legitimate owner of the trigger. Either way, this is an example of data tampering and a loss of data integrity.

Note that there are relatively few standards or best practices for IoT security design and testing, although some related guidance is being developed by the Cyber-Physical Systems Public Working Group [NIST 2014a] and in documents such as the National Institute of Standards and Technology's *Guidelines for Smart Grid Cybersecurity* [NIST 2014b].

6 Summary

This document offers an underlying and foundational science for IoT-based technologies on the realization that IoT involves *sensing*, *computing*, *communication*, and *actuation*. We presented a common vocabulary to foster a better understanding of IoT and better communication between those parties discussing IoT. We acknowledge that the Internet is a network of networks, but we believe that focusing on restricted NoTs in a bounded way gives better traction to addressing *trustworthiness* problems that an unbounded Internet does not. Some may argue that every 'thing' in a NoT is ultimately a service. The primitives definitely offer services, however because of the combinatorics of mixtures of hardware and software 'things', and we prefer to distinguish.

Five primitives and six elements that impact IoT trustworthiness have been presented. Primitives are the building blocks; elements are the less tangible *trust* factors impacting NoTs. Primitives also allow for analytics and formal arguments of IoT use case scenarios. Without an actionable and universally-accepted definition for IoT, the model and vocabulary presented here expresses how IoT, in the broad sense, *behaves*.

Use case scenarios employing the primitives afford us quicker recommendations and guidance concerning a NoT's potential trustworthiness. For example, authentication can be used in addressing issues such as geo-location and sensor ownership, but authentication may not be relevant if an adversary owns the sensors and can obtain that information based on proximity. Encryption can protect sensor data transmission integrity and confidentiality including cloud-to-cloud communication, but it might render the IoT sensors unusable due to excessive energy requirements. While fault-tolerant techniques can alleviate reliability concerns associated with inexpensive, replaceable, and defective third party 'things', they can also be insecure and induce communication overhead and increased attack surfaces. In short, primitives and how they can be composed create a design vocabulary for how to apply existing technologies that support IoT trustworthiness.

These primitives are simply *objects* with *attributes*. The five, along with the context offered by the six elements, form a design *catalog* for those persons and organizations interested in exploring and implementing current and future IoT-based technology.

Appendix A—References

[Kuhn 2016] D. R. Kuhn, V. Hu, D. F. Ferraiolo, R. N. Kacker, and Y. Lei, "Pseudo-exhaustive Testing of Attribute Based Access Control Rules," *International Workshop on Combinatorial Testing* at the *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Chicago, Illinois, April 10-15, 2016.

[Li 2004] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, Hong Kong, March 7-11, 2004, pp. 564-574. http://dx.doi.org/10.1109/INFCOM.2004.1354528.

[NIST 2015] M. Weiss, J. Eidson, C. Barry, D. Broman, L. Goldin, B. Iannucci, E. A. Lee, and K. Stanton, *Time-Aware Applications, Computers, and Communication Systems (TAACCS)*, NIST Technical Note (TN) 1867, National Institute of Standards and Technology, Gaithersburg, Maryland, February 2015, 26pp. http://dx.doi.org/10.6028/NIST.TN.1867.

[Voas 1993] J. M. Voas and K. W. Miller, "Semantic metrics for software testability," *Journal of Systems and Software* vol. 20, no. 3 (March 1993), pp. 207-216. http://dx.doi.org/10.1016/0164-1212(93)90064-5.

[Voas 1995] J. M. Voas and K. W. Miller, "Software testability: the new verification," *IEEE Software* vol. 12, no. 3 (May 1995), pp. 17-28. http://dx.doi.org/10.1109/52.382180.

[NIST 2014a] *Cyber-Physical Systems Public Working Group Workshop* [Web page], National Institute of Standards and Technology, August 11-12, 2014. http://www.nist.gov/cps/cps-pwg-workshop.cfm [accessed 7/13/16].

[NIST 2014b] The Smart Grid Interoperability Panel—Smart Grid Cybersecurity Committee, *Guidelines for Smart Grid Cybersecurity*, NIST Interagency Report (NISTIR) 7628 Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland, September 2014, 668pp. http://dx.doi.org/10.6018/NIST.IR.7628r1.

Appendix B—Acronym Glossary

AC	Alternating Current
AI	Artificial Intelligence
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
DDoS	Distributed denial-of-service attack
FPGA	Field-programmable gate array
IEC	International Electrotechnical Commission
IEEE	Institute of Electronics and Electrical Engineers
IoT	Internet of Things
ISO	International Standards Organization
MAC	Media access control
NIST	National Institute of Standards and Technology
NoT	Network of Things
PHM	Prognostics and Health Management
RFID	Radio Frequency Identification
SDO	Standards Developing Organization
USB	Universal Serial Bus
Wi-Fi	Any wireless local area network product based on IEEE 802.11

Appendix C—Additional Takeaway Messages

1. 'Things' may be all software, hardware, a combination of both, and human.
2. A NoT may or may not employ 'things' connected to the Internet.
3. The number of 'things' in a NoT fuels functional complexity and diminishes *testability* unless *observability* is boosted by internal test instrumentation (assertions).
4. NoTs bound scalability and complexity, and therefore enhance arguments for trustworthiness since assurance techniques generally offer better efficacy to less complex systems.
5. Known threats from previous genres of complex software-centric systems apply to NoTs.
6. Security flaws and threats in NoTs may be exacerbated by the composition of 3rd party 'things.' This creates an *emergent* class of security 'unknowns.'
7. NoTs may have the ability to self-organize, self-modify, and self-repair when artificial intelligence (AI) technologies are introduced, e.g., neural networks, genetic algorithms, and machine learning. If true, NoTs could potentially rewire their security policy mechanisms and implementations, or disengage them altogether.
8. "After the fact" forensics for millions of composed, heterogeneous 'things', is almost certainly not possible in linear time.
9. 'Things' will be heterogeneous. Counterfeiting of 'things' may lead to seemingly non-deterministic behavior making testing's results appear chaotic. Counterfeit 'things' may lead to illegitimate NoTs.
10. Properly *authenticating* sensors may be a data integrity risk, e.g., the 'who is who' question. 'Things' may deliberately misidentify themselves.
11. 'Things' may be granted a nefarious and stealth connection capability, that is, coming and going in instantaneous time snapshots, leaving zero *traceability*. This is a "drop and run" mode for pushing external data into a NoT's workflow. This may be mitigatable via authentication, cryptography, and possibly others. "Drop and run" affects trustworthiness.
12. *Actuators* are 'things.' If they are fed malicious data from other 'things', issues with life-threatening consequences are possible if the actuator operates in a safety-critical environment.
13. NoTs have workflows and dataflows that are highly *time*-sensitive – NoTs need communication and computation *synchronization*. Defective local/global clocks (timing failures) lead to deadlock, race conditions, and other classes of system-wide, NoT failures.