INTERNET OF THINGS:
ROBUSTNESS AND RELIABILITY

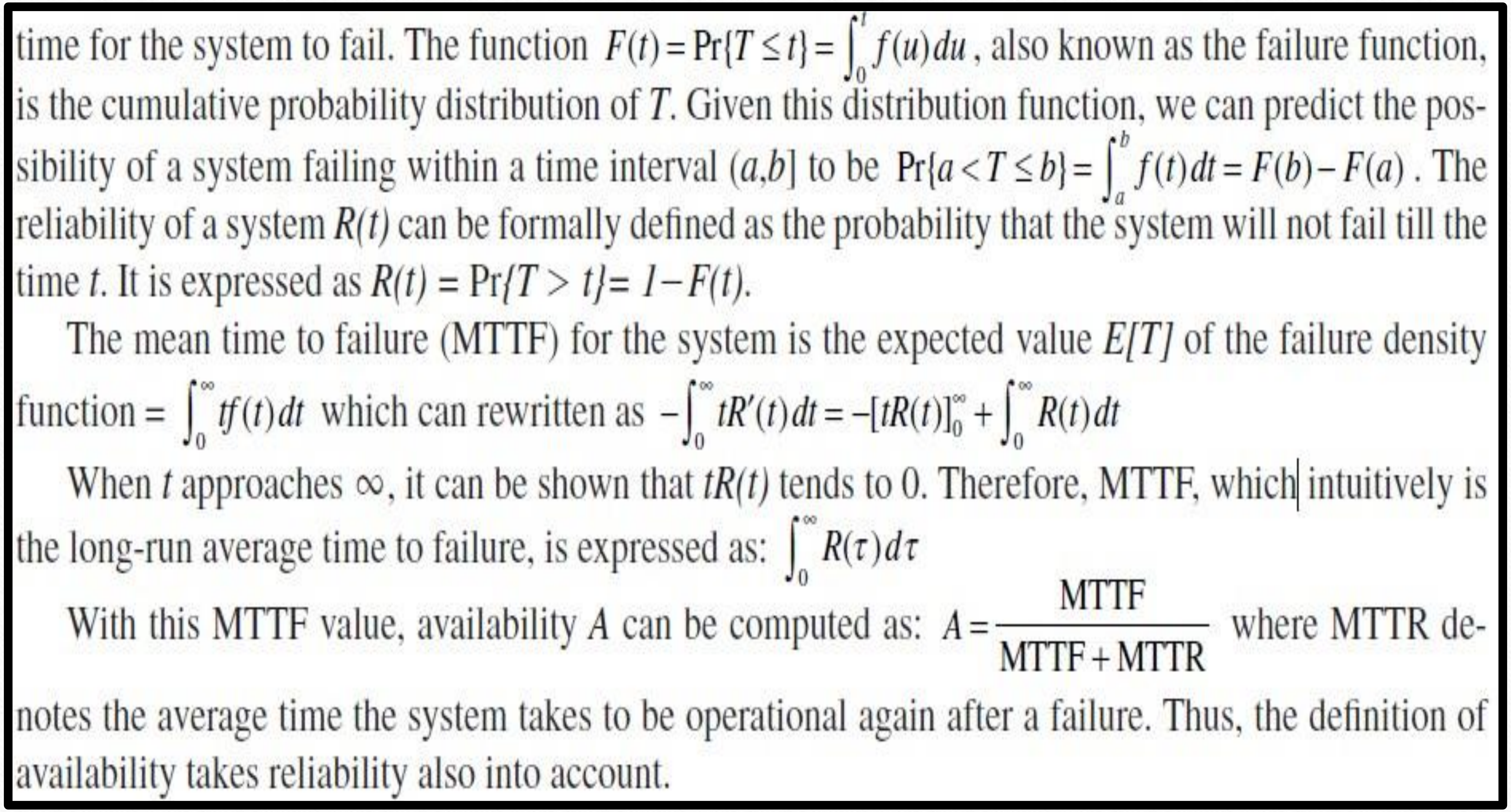
INTRODUCTION

- Building a reliable computing system has always been an important requirement for the business and the scientific community.
- By the term reliability, we mean how long a system can operate without any failure.
- Along with reliability, there is another closely related quality attribute, called availability.
- Informally, availability → percentage of time that a system is operational to the user.
- An internet of things (IoT) system deploys a massive number of network aware devices in a dynamic, error-prone, and unpredictable environment, and is expected to run for a long time without failure.
- To commission such a system and to keep it operational, it is therefore essential that the system is designed to be reliable and available.

INTRODUCTION (cntd).

- Since the exact time of a failure of any operational system is not known a priori, it is appropriate to model the time for a system to fail as a (continuous) random variable.
- Let $f(t)$ → failure probability density function, which denotes the instantaneous likelihood that the system fails at time t.
- Next, we would like to know the probability that the system will fail within a time t, denoted by $F(t)$.
- Let T → Time for the system to fail.

INTRODUCTION (cntd).



time for the system to fail. The function $F(t) = \Pr\{T \le t\} = \int_0^t f(u)\,du$, also known as the failure function, is the cumulative probability distribution of T. Given this distribution function, we can predict the possibility of a system failing within a time interval $(a,b]$ to be $\Pr\{a < T \le b\} = \int_a^b f(t)\,dt = F(b) - F(a)$. The reliability of a system $R(t)$ can be formally defined as the probability that the system will not fail till the time t. It is expressed as $R(t) = \Pr\{T > t\} = 1 - F(t)$.

The mean time to failure (MTTF) for the system is the expected value $E[T]$ of the failure density function = $\int_0^\infty t f(t)\,dt$ which can rewritten as $-\int_0^\infty tR'(t)\,dt = -[tR(t)]_0^\infty + \int_0^\infty R(t)\,dt$

When t approaches ∞, it can be shown that $tR(t)$ tends to 0. Therefore, MTTF, which intuitively is the long-run average time to failure, is expressed as: $\int_0^\infty R(\tau)\,d\tau$

With this MTTF value, availability A can be computed as: $A = \dfrac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$ where MTTR denotes the average time the system takes to be operational again after a failure. Thus, the definition of availability takes reliability also into account.

INTRODUCTION (cntd).

- Availability has been one of the most important quality attributes to measure the extent of uninterrupted service that a distributed and more recently a cloud-based system provides.
- It has also been an important metric to define the Service Level Agreement (SLA) between the service provider (a SaaS or an IaaS provider) and the service consumer.

INTRODUCTION (cntd).

- From the definition, it is obvious that a system which is highly reliable (high MTTF) will tend to be highly available as well.
- However, the mean time to recover or MTTR brings another alternative means to achieve high availability.
- One can design a highly available system even with components having relatively poor reliability (not very large MTTF), provided that the system takes a very little time to recover when it fails.
- Although the hardware industry has always strived to make the infrastructure reliable (ie, increase MTTF), today it has possibly reached its limit.
- Increasing MTTF beyond a certain point is extremely costly, and sometimes impossible.

INTRODUCTION (cntd).

- In view of this, it becomes quite relevant to design a system equipped with faster recovery mechanisms.
- This observation has led to the emergence of recovery oriented computing (ROC) paradigm, which has now been considered to be a more cost-effective approach to ensure the service continuity for distributed and cloud-based systems.
- The fundamental principle of ROC is to make MTTR as small as possible.
- For an IoT-based system, the participating components can have high failure possibilities.
- In order to ensure that an IoT system always remains operational, the ROC becomes an attractive and feasible approach.

INTRODUCTION (cntd).

- Along with **reliability** and **availability**, the term **serviceability** coined by is frequently used to indicate the ease with which the deployed system can be repaired and brought back to operation.

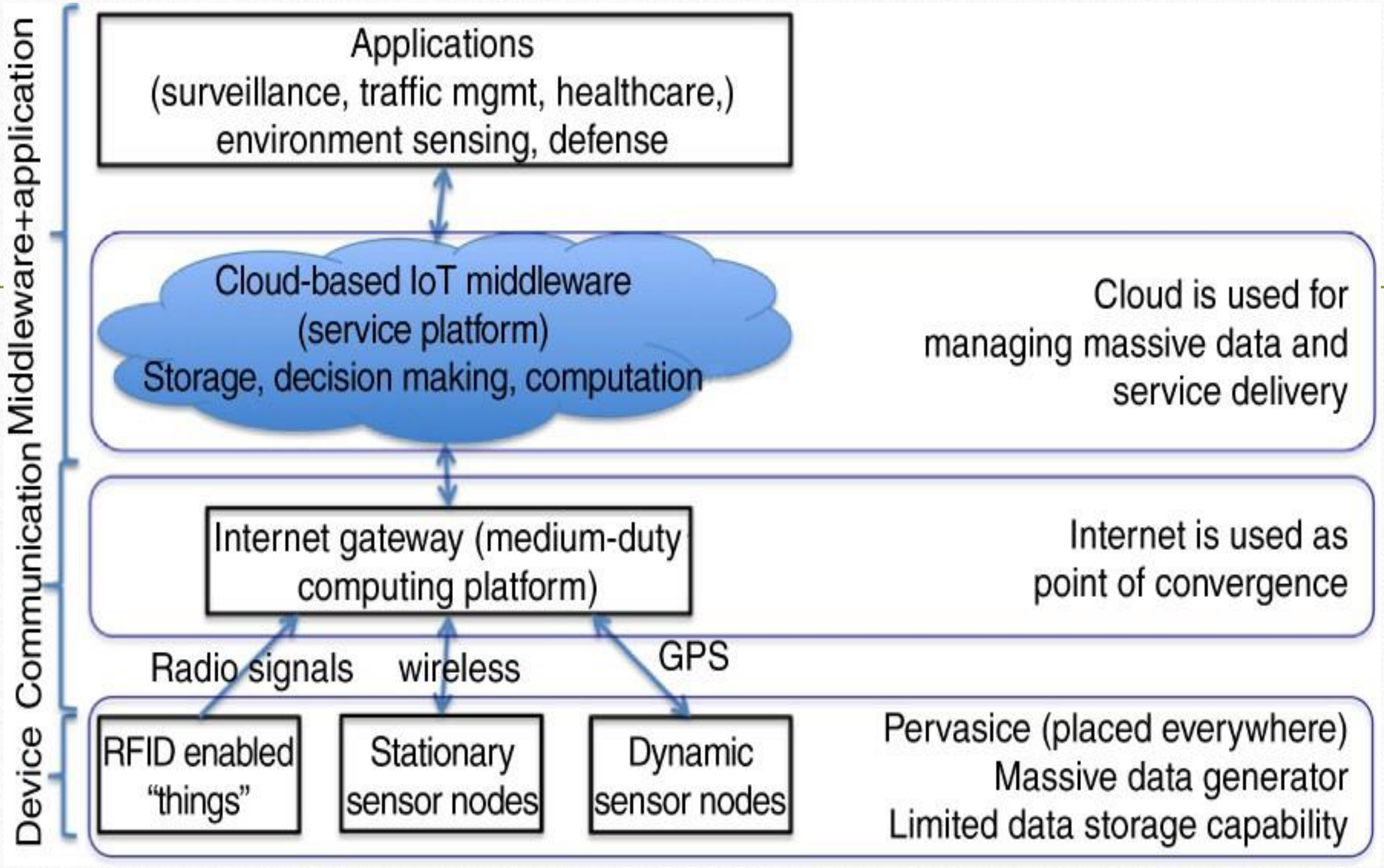
- Thus, Serviceability → reduction of the MTTR using various failure-prevention methods (prediction, preventive maintenance), failure detection techniques (through monitoring), and failure handling approaches (by masking the impact of an error, recovery).

- The goal of serviceability is obviously to have a zero repair time, thereby achieving a near 100% availability.

IOT CHARACTERISTICS AND RELIABILITY ISSUES

- With the advancement in infrastructure and wireless communication, proliferation of new communication aware devices of various form factors, and with the introduction of cloud computing paradigm, Internet of Things-based applications are emerging.
- Such an application, distributed on multitude of devices, is more embedded into the business environment than ever before.
- For instance, smart, network accessible cameras can be placed at strategic locations in a cluster of buildings or on streets, smart meters may be installed in a power-grid, tiny embedded devices can be used for health monitoring, vehicles in a city can be equipped with GPS-based sensors, and static wireless sensors can be embedded in modern appliances like a television or a refrigerator.
- These network-enabled devices can run distributed processes, which in turn can coordinate, exchange data, and take critical decisions in real time.
- Such a system is expected to be deployed once and be operational forever.

IOT ARCHITECTURE IN BRIEF - LAYERED ARCHITECTURE REFERENCE MODEL


Device Communication Middleware+application
Applications
(surveillance, traffic mgmt, healthcare,)
environment sensing, defense
Cloud-based IoT middleware
(service platform)
Storage, decision making, computation
Cloud is used for
managing massive data and
service delivery
Internet gateway (medium-duty
computing platform)
Internet is used as
point of convergence
Radio signals
wireless
GPS
RFID enabled
"things"
Stationary
sensor nodes
Dynamic
sensor nodes
Pervasice (placed everywhere)
Massive data generator
Limited data storage capability

Iot Architecture In Brief:
- DEVICE LAYER

- The lowest layer consists of devices with a low to moderate compute and communication capability.
- These devices are typically battery operated, and can execute Tiny OS like RIOT or Contiki.
- They typically receive data from the environment, perform local processing on the data, and then transmit the result.

IOT ARCHITECTURE IN BRIEF- COMMUNICATION LAYER

- The devices transmit data through WIFI, GSM/GPRS, Bluetooth, and radio frequency for RFID-enabled devices.
- The communication layer comprises devices (routers, signal transceivers, etc.) that are responsible for reliable transmission of data.

IOT ARCHITECTURE IN BRIEF- APPLICATION LAYER

- In this layer, an IoT middleware resides, with which the devices interact to exchange data.
- We refer to this middleware as the “service platform”.
- The service platform can be hosted on the cloud to exploit the on-demand and scalable infrastructure capability of the cloud.
- In order to make the application layer flexible and extendible, well-known architecture patterns like hub and spoke, microkernel, and blackboard-based design can be adopted.
- The smart devices can join the network on-demand, and the platform, acting as a hub, processes a massive amount of data from the network of devices.

IOT ARCHITECTURE IN BRIEF- APPLICATION LAYER (CONTINUED)

- On top of this middleware, the IoT specific application can reside which can perform various on and offline tasks which can be low-latency real-time responses or heavy-duty data analysis activities.
- Such an approach is beneficial from two perspectives.
- For the cloud, the overall reach of the system becomes far more deep as the real-world devices get embedded in the application domain.
- For the devices, their limited compute and storage capabilities are compensated by virtually unlimited resources in the cloud, and the cloud becomes a point of convergence for the devices.

DIFFERENT CATEGORIES OF APPLICATIONS

We can classify IoT applications into the following categories from reliability and availability perspective:

- Zero tolerance
- Restartable
- Error Tolerant

1. ZERO TOLERANCE

- When an IoT system is placed in a mission critical scenario, specifically in the health care domain, where network aware devices monitor the health of the patient, or a network-enabled pacemaker device that interacts with a larger health-care platform, the system components do not tolerate any failure during its mission time (when it is actively working).
- In other words, the MTTF(which impacts reliability) for these components should be strictly greater than the mission time.
- Furthermore, the MTTR for such a system should be close to zero during the mission time.

2. RESTARTABLE

- **Here the IoT system can tolerate a faulty component or event the entire system (though undesirable) to restart without any catastrophic impact.**
- **For instance, an IoT system for urban transport, having components embedded in vehicles, can afford to restart, if the embedded component fails.**
- **Here, more than MTTF being high, the goal is to make MTTR as small as possible.**

3. ERROR TOLERANT

- Here the nature of the application is such that a part of the system can tolerate the erroneous input for some time, within the user-defined safety limit before getting it fixed.
- For instance, an unmanned surveillance system providing real-time routine information of an agricultural land, can afford to send poor/incorrect data before it is rectified.
- Similarly, a recommendation system of the next generation ecommerce system reading a real-time input data-feed can accept erroneous data (and obviously generate erroneous recommendations) for a small time before the error conditions are rectified.

FAILURE SCENARIOS

- Like any mission critical system, we say that an IoT-based system becomes unreliable or unavailable when the system either fails to respond to a request or provides an unexpected, incorrect service.
- A service failure happens when faults are not handled properly.
- Researchers and practitioners have extensively studied various faults and remedial actions to keep software operational in a business critical scenario.

FAILURE SCENARIOS (CONTINUED)

- Broadly, these faults are categorized as (i) development fault (ii) infrastructure fault, and (iii) interaction faults.
- **Development faults** are induced by incorrect implementation of the software, whereas **infrastructure faults** are caused due to unanticipated faults in the hardware.
- **Interaction faults** occur due to interaction with other software modules or incorrect data format.
- Let us analyze how they are relevant in the IoT context.

1. INFRASTRUCTURE FAULT

- The cluster of network-enabled devices in an IoT-based system are expected to operate in unanticipated scenarios.

Some of these scenarios can lead to infrastructure failures. For instance:

1. In a given IoT application scenario, the network-enabled devices ought to be embedded in a specific environment to gather and process data stream. The devices can fail due to the physical condition and interference with the environment in which they operate. Such an operating condition can reduce the life of such devices drastically due to the physical deterioration.

1. INFRASTRUCTURE FAULT (CONTINUED)

- Consider the scenario of electronic tracking of animals, which is an important business problem in Norway.
- The sensor attached on an animal has an extremely high chance of a failure, resulting in poor or erroneous data transmission, or a complete data loss.
- This may not result in an imminent failure of the service platform but it can certainly lead to corruption of data, incorrect interpretation, and an eventual failure of service.

1. INFRASTRUCTURE FAULT (CONTINUED)

2.The external environment may provide unexpected inputs to IoT entities resulting in a computation failure in the device.

3.The processors of these devices have been designed keeping a small form-factor in mind rather than making them highly fault-tolerant. Thus these devices can be much more fault-prone than a normal computer.

4.Many a times these devices run on a battery which can severely limit their compute time and can cause unexpected termination of a computation.

The overall reliability and availability of the system will obviously depend on the extent to which these devices can withstand these unexpected scenarios.

2. INTERACTION FAULT

- Network-enabled devices and appliances have widely varying compute capabilities.
- When these devices are made to communicate with each other and share data, there can be operational failures due to several reasons:

2. INTERACTION FAULT (CONTINUED)

1. The entire network of the communication components can fail. Consider the same scenario of electronic tracking of animals. Now consider a likely situation where the GPRS backbone fails in the IoT system that is supposed to be used to report the animal tracking data. Such an event can certainly impact the overall system functionality.

2. INTERACTION FAULT (CONTINUED)

2. Due to the heterogeneity of the devices, there could be an “impedance mismatch” of the data being exchanged. Such a scenario can occur when the IoT system allows a device to join the network in real time.

- An example of such a scenario is the management of vehicular network in a city where the vehicles joining dynamically may not comply with the protocol. In such a situation, it will not be possible to interpret the data and take the appropriate action.

INTERACTION FAULT (CONTINUED)

3. Interaction faults are also caused by unexpected workload coming from various IoT components.

3. FAULT IN SERVICE PLATFORM

- Consider the above diagram of architecture reference model where the service platform acts as a hub that collects data from various network aware sensor objects and processes the data.
- It is unlikely that the platform will be built from the scratch.
- It will integrate many third party products and will be integrated with external partner systems.
- Even if we assume that this middleware has been thoroughly tested for its own functionality, many transient faults can be due to off-the-shelf components

3. FAULT IN SERVICE PLATFORM (CONTINUED)

- The reliability of these third party components may be questionable and can often be a cause of failure of the main system.
- Additionally, the external partner systems of the IoT application middleware can fail or provide incorrect data, which can result in a failure of the middleware.
- When the platform fails due to these faults, it will not be able to process the incoming messages and route them.

RELIABILITY CHALLENGES: - **Making Service Available To User**

- The aim of an IoT application is to provide an immersive service experience through a tightly coupled human–device interaction in real time.
- Therefore, it is highly important that the availability of the system be judged from the user perspective.
- This is known as "user perceived availability" where the perceived availability is about delivering the service to the user, not just surviving through a failure.

RELIABILITY CHALLENGES: - **Making Service Available To User**

- A relatively old study on Windows server shows that though the server was available for 99% of the time (obtained from the server log), but the user perceived it to be just 92%.
- To improve the user perceived availability, the IoT service platform has an important role to play where it needs to ensure that the user service requests (coming from the application layer on top of the middleware as shown in diagram) are always responded to even in extreme circumstances.

RELIABILITY CHALLENGES :-
Serviceability of IoT System

- It is quite likely for an IoT system to have a set of devices that can dynamically come and join the system.
- In such a case, ensuring serviceability without disrupting the ongoing activities is more difficult in the case of IoT devices.
- In a traditional high-available system, nodes as well as the software running on them go through a scheduled maintenance when the software and patches are upgraded to prevent any upcoming failure.
- Such a traditional maintenance may not be feasible for an IoT system.
- For instance, a device is located in a mission critical location where one cannot simply perform a shutdown (for instance, smart healthcare devices like network aware pacemaker).

RELIABILITY CHALLENGES – Serviceability Of Iot System (Cntd.)

- The devices may operate on a relatively low network bandwidth and on a limited battery power where over-the-wire large data (such as a software patch) transmission may not be practical all the time.
- For this reason, the software, protocols, and applications that are created in the IoT framework need to be tested not only for functionality, and traditional nonfunctional features, but also for their fault-tolerance so that it can remain operational for a longer duration without any repair.

RELIABILITY CHALLENGES:-

Reliability At Network Level

- Most Internet of Things applications for buildings, factories, hospitals, or the power grid are long-term investments that must also be operable for a long time.
- The networks can also be unmanaged (eg, home automation, transport applications).
- This implies that the network must be able to configure itself as environmental conditions or components in the network itself change so that the information can always be transmitted from one application to the other reliably.
- There can be further complexities in using sensor devices.

RELIABILITY CHALLENGES:-

Reliability At Network Level(Cntd.)

- The links used in most sensor networks today use completely unregulated bands of frequency.
- As a result, it is very easy that the signals from a sensor device interfere with another and make the links unreliable.
- For instance, if a newly deployed IoT sensor network starts using the same channel as someone's existing WLAN access point, the interference can disable critical sensor data reporting.
- Links in sensor networks are often more unreliable than the Internet due to the lack of regulation.
- Therefore, it is highly desirable to have some form of reliable transport protocol for IoT that is as power-saving as UDP and as reliable as TCP.

RELIABILITY CHALLENGES :–

Device Level Reliability

- From the network level, let us now focus on the embedded devices that are connected via network.
- Even when the network is reliable, there are scenarios when the applications running on these devices may generate poor quality data, which makes the entire computation unreliable.
- Consider a scenario when the device needs to gather image from the physical world where it is embedded.
- Due to the environmental condition, the quality of the images captured can be below the acceptable level; as a result the associated inferences drawn from the captured images become unreliable.
- For sensor devices, the environmental condition can result in a bit error.

RELIABILITY CHALLENGES :–

Device Level Reliability(Cntd.)

- Computing devices are now being deployed in medical monitoring and diagnostic systems.
- Such a system that not only performs monitoring but also provides recommendation to the physicians, is safety critical.
- Similarly devices used in fire safety scenarios need to be zero tolerant.
- However, a sensor device embedded in a fire alarm system also has a high chance of malfunctioning during the critical time.
- The reliability challenge in this case is to ensure the timely diagnosis and alert generation during the critical time, even when some part of the system (for instance, the fire sensor) malfunctions or sends poor quality information.

RELIABILITY CHALLENGES :–
Device Level Reliability(Cntd.)

- The security threats of these safety critical systems can adversely impact reliability to a large extent.
- Smart mobile phones of today have a good amount of computing capability, and they are becoming an intrinsic part of IoT applications.
- For cell-phone-based communication, it is the inexpensive and power constrained mobile phone that poses the reliability challenges than the communication infrastructure.

PRIVACY AND RELIABILITY

- Data privacy is an important part of IoT, specifically when an IoT system allows a machine-to-machine interaction, where machines can join the network dynamically.
- In this context, identity management, and proving identity on-demand has been considered an important mechanism to ensure the authenticity of the communicating parties.
- For instance consider an IoT-based vehicle management system, which expects vehicles to reveal their identity in a vehicular network.
- The system can create an alarm and can trigger actions if the deployed sensors on a street sense that a car has not revealed the identity.

PRIVACY AND RELIABILITY (Cntd.)

- However, this alarm can be a false one if the car is a police car, which can reveal its identity to another police car and to the designated staff at the police station, but keep its identity hidden during undercover work otherwise.
- Under such a scenario, the real-time surveillance system's reliability of detecting intrusion becomes questionable due to the inaccuracy.

INTEROPERABILITY OF DEVICES

- Since IoT allows heterogeneity of devices interacting with each other, there is always a possibility that the participating devices cannot exchange information due to the lack of standardization.
- As of today, the standardization of the communication among the devices has not been enforced in IoT.
- Traditionally, system reliability is often associated with various other quality
  - attributes like performance, availability, and security.

INTEROPERABILITY OF DEVICES (cntd.)

- Interoperability, an important quality attribute by itself, has not traditionally been considered in conjunction with reliability.
- However, with the emergence of IoT, we now find that if an IoT infrastructure has devices that are not interoperable, and if the IoT system requires that the devices can come and join dynamically, the overall reliability of the infrastructure to perform the intended service is bound to suffer significantly unless the dynamically participating devices are interoperable.

RELIABILITY ISSUES DUE TO ENERGY CONSTRAINT

- Autonomous devices of an IoT system such as automated surveillance system need to collect and process data in real time from the environment for a long duration.
- The data stream is transmitted from a set of embedded sensors, which are running on battery power.
- Even in an ideal scenario, when the environmental condition does not interfere with the functioning of the sensors, the reliability of the overall system can still be impacted due to the limited power supply.
- It is therefore essential that the IoT infrastructure ensures both reliability and low energy consumption [also referred to as Energy Efficient Reliability (EER) in the literature.

ADDRESSING RELIABILITY

- Traditionally, the designers of a reliable system take the approach of either fault prevention or fault tolerance to ensure reliability and availability.
- To prevent a failure, proactive actions are taken to stop the imminent failure of a running system.
- In the case of an IoT-based system, prevention many a times becomes quite difficult, as there are many unanticipated external as well as transient faults due to reasons often beyond the realm of our control.
- Fault-tolerance on the other hand implies that the system is able to operate in the presence of faults.

ADDRESSING RELIABILITY (CNTD.)

- Fault-tolerance is achieved mainly through nullifying the impact of an error (*also known as error masking*), *error detection*, *and recovery*.
- The system can nullify the impact of an error mostly through employing a set of redundant components — expecting that while the primary component fails due to an unanticipated error, the other clones still survive the impact to render the service.

ADDRESSING RELIABILITY (CNTD.)

- ROC, essentially consists of detecting an error early, taking the corrective actions, and performing a speedy recovery.

- A high-level guideline of making an IoT system fault-tolerant.

- Three broad guidelines for sensor objects namely

i. making the objects participating in the IoT network robust, by default .

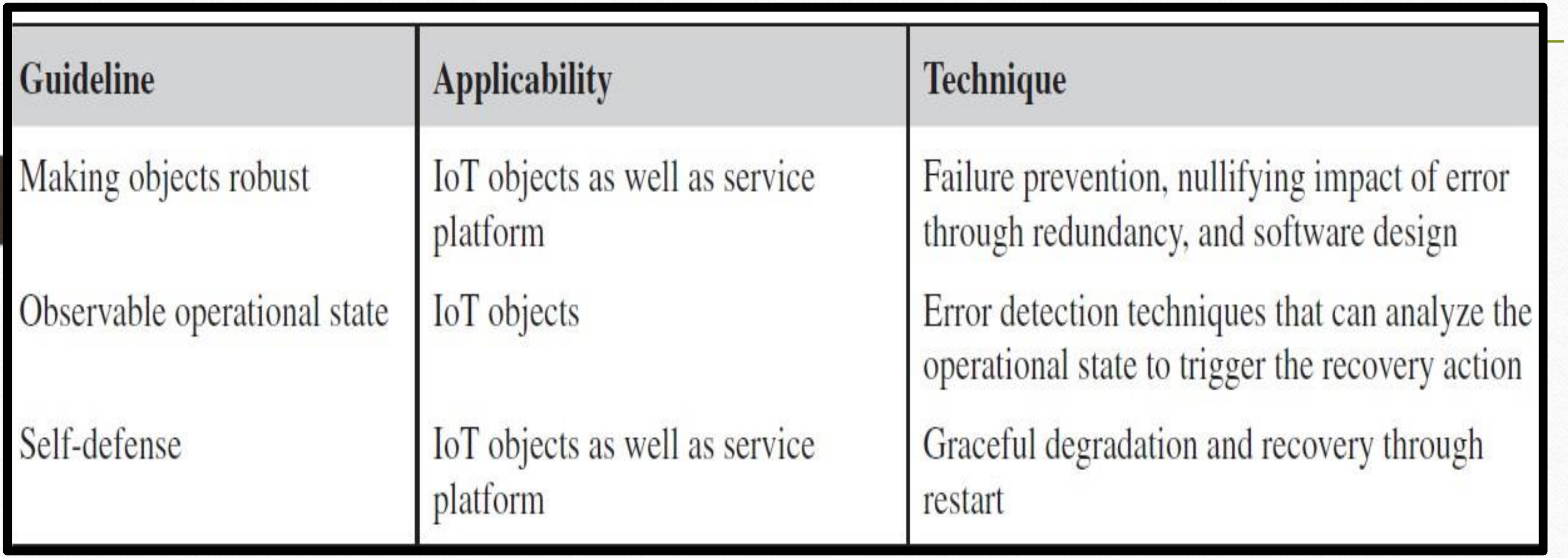
ii. build capability to make the operational state of the objects observable, and

iii. incorporate self- defense and recovery mechanism in the participating objects.

- These guidelines are not only applicable for the sensor objects, but also for the service platform, as well as the communication infrastructure.

ADDRESSING RELIABILITY (CNTD.)

Table: summarizes the applicability:

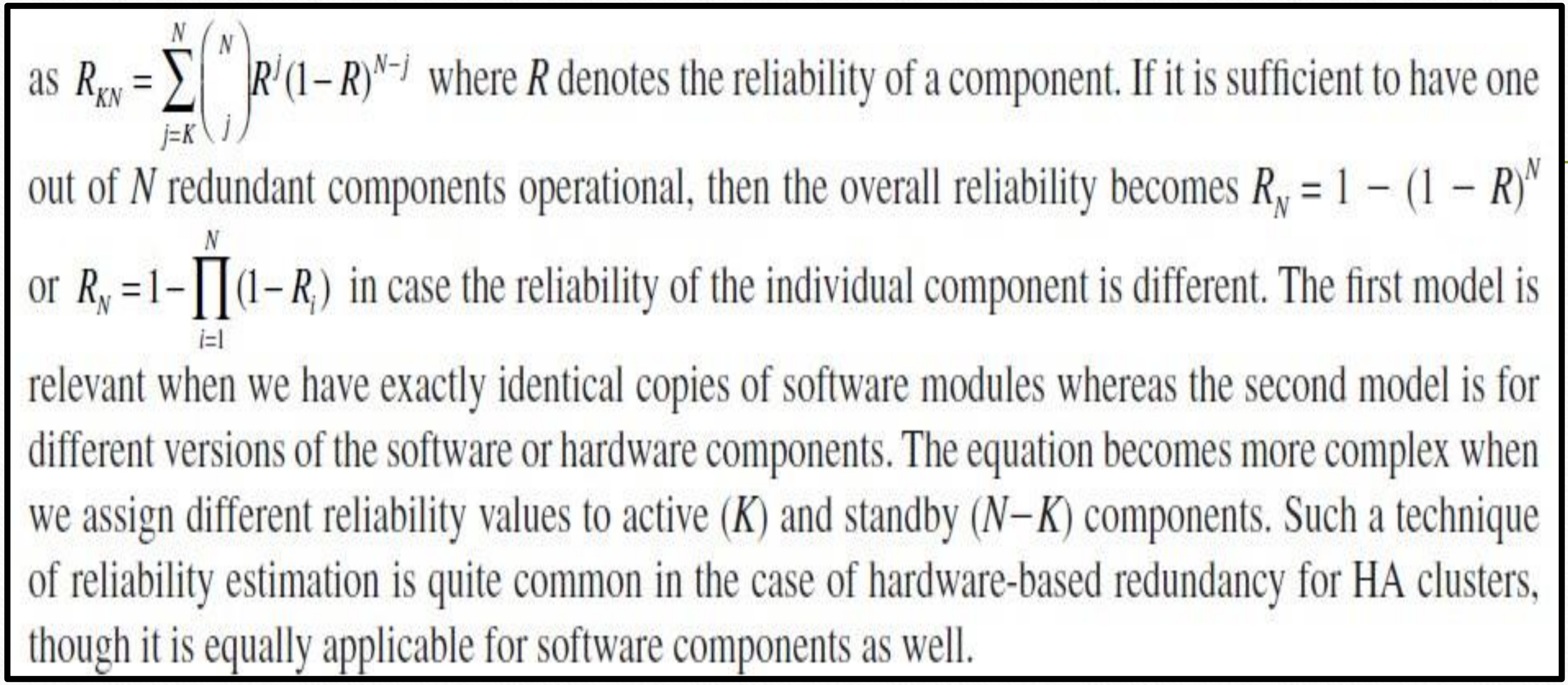


Guideline	Applicability	Technique
Making objects robust	IoT objects as well as service platform	Failure prevention, nullifying impact of error through redundancy, and software design
Observable operational state	IoT objects	Error detection techniques that can analyze the operational state to trigger the recovery action
Self-defense	IoT objects as well as service platform	Graceful degradation and recovery through restart

NULLIFYING IMPACT OF FAULT

- A common approach for a traditional high-availability cluster is to reduce the impact of a fault as much as possible by eliminating the single point of failure (SPOF) both at the hardware and at the software level.
- A common technique for this is to increase redundancy at various levels such as the infrastructure, the network, and in the software.
- A simplified model to calculate the reliability of a system with redundancy having N components (a hardware or a software), of which it is sufficient to have K components operational, can be expressed

NULLIFYING IMPACT OF FAULT(Cntd.)



as $R_{KN} = \sum_{j=K}^{N} \binom{N}{j} R^j (1-R)^{N-j}$ where R denotes the reliability of a component. If it is sufficient to have one out of N redundant components operational, then the overall reliability becomes $R_N = 1 - (1 - R)^N$ or $R_N = 1 - \prod_{i=1}^{N} (1 - R_i)$ in case the reliability of the individual component is different. The first model is relevant when we have exactly identical copies of software modules whereas the second model is for different versions of the software or hardware components. The equation becomes more complex when we assign different reliability values to active (K) and standby ($N-K$) components. Such a technique of reliability estimation is quite common in the case of hardware-based redundancy for HA clusters, though it is equally applicable for software components as well.

REDUNDANCY IN SERVICE PLATFORM DESIGN

- A good design to build a robust IoT service platform is to introduce a set of loosely coupled components so that the failure of one component does not bring the entire platform down.
- Depending on the application scenario, one can then deploy multiple copies of the critical components.
- Many times database systems become an integral part of the platform where the data from different communicating objects are collected for analysis.

REDUNDANCY IN SERVICE PLATFORM DESIGN (cntd.)

- In such a case, the database system can also be a SPOF.

Depending on the nature of the application one can adopt the following Redundancy Strategies:

1. Load balancing across multiple databases: The load of the incoming data can be shared across multiple databases
2. Data replication: Multiple copies of data can be stored in the vast storage areas of clouds.

REDUNDANCY IN M2M TOPOLOGY

- In some application domains such as a **large scale environment monitoring** system, the **IoT devices can be scattered** and **decentralized.**
- ~~Unlike a hub and spoke model, these devices may require to interact~~ with each other to **exchange** their **local computation.**
- In order to increase the redundancy in such a scenario, researchers have suggested **storing replicas of the local data** and **computation on the neighboring devices** so that a device **failure does not result in** a **data loss.**
- A **fault tolerant middleware** has been proposed where the applications (called services) are replicated across multiple nodes, with one being active at a time.

REDUNDANCY IN M2M TOPOLOGY(cntd.)

- Each device computes its own application, and also knows where its own local applications have been replicated.
- Furthermore, it monitors another device(s) through a heartbeat mechanism.
- For this purpose, the device maintains a copy of the application deployment information of the device it is monitoring.
- This topology ensures that there is no SPOF for any application.
- The failures are detected in a decentralized manner and failure recovery is performed autonomously.

GRACEFUL DEGRADATION

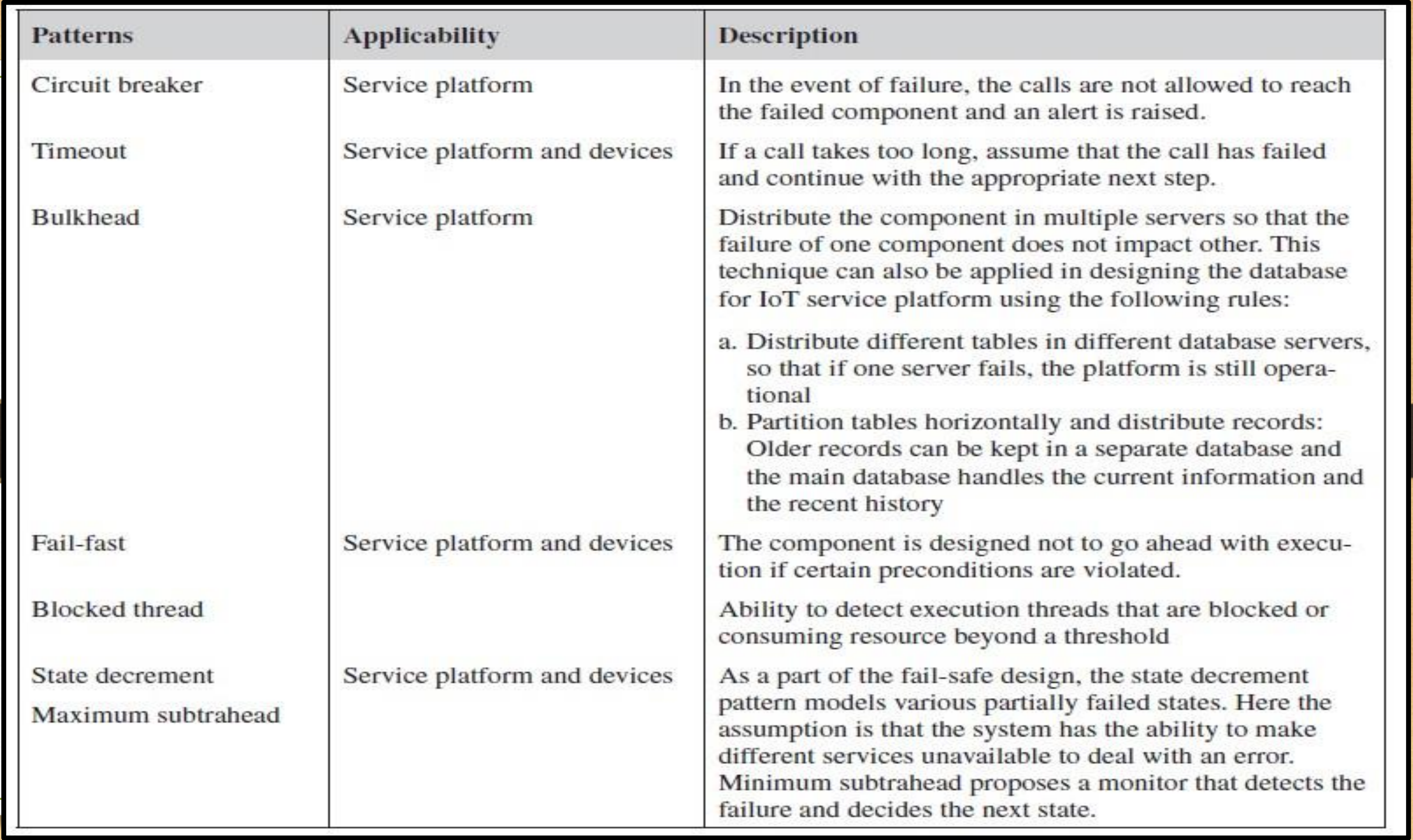
- In order to make the autonomous objects as well as the service platform of an IoT system survive for long, a graceful degradation (also known as fail-safe) capability should be built-into the components.
- When an unexpected event occurs, a gracefully degradable system can remain partly operational, rather than failing completely.
- However, as the system designer, it is important to define what degradation means in a given application context.

GRACEFUL DEGRADATION (cntd.)

- For instance, in an autonomous power-grid management system, the monitoring system can selectively stop power transmission to a certain region keeping the main controlling unit unimpacted.
- In the case of office automation system, the elevators can run at a lower speed when the system falls back to a degraded mode.
- For a component-based software system, a fault in a particular component can result in nonavailability of the service provided by the failed component rather than bringing the entire system down.

GRACEFUL DEGRADATION :- SOFTWARE DESIGN

- Building a gracefully degradable system requires careful planning and design.
- SaaS platforms today are building applications that are failure resilient.
- Some of these design patterns can be applied in the IoT context



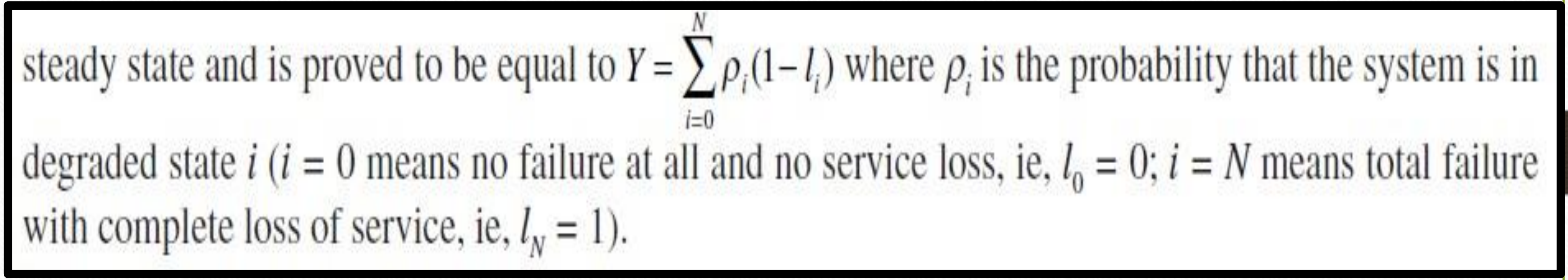
Patterns	Applicability	Description
Circuit breaker	Service platform	In the event of failure, the calls are not allowed to reach the failed component and an alert is raised.
Timeout	Service platform and devices	If a call takes too long, assume that the call has failed and continue with the appropriate next step.
Bulkhead	Service platform	Distribute the component in multiple servers so that the failure of one component does not impact other. This technique can also be applied in designing the database for IoT service platform using the following rules: a. Distribute different tables in different database servers, so that if one server fails, the platform is still operational b. Partition tables horizontally and distribute records: Older records can be kept in a separate database and the main database handles the current information and the recent history
Fail-fast	Service platform and devices	The component is designed not to go ahead with execution if certain preconditions are violated.
Blocked thread		Ability to detect execution threads that are blocked or consuming resource beyond a threshold
State decrement	Service platform and devices	As a part of the fail-safe design, the state decrement pattern models various partially failed states. Here the assumption is that the system has the ability to make different services unavailable to deal with an error. Minimum subtrahead proposes a monitor that detects the failure and decides the next state.
Maximum subtrahead		

Graceful Degradation :-

Performability Model

- A well-known technique to evaluate the graceful degradation, specifically the State decrement pattern, is to build a performability model of the system.
- A Markov model to capture the steady degradation from no-failure state to total failure situation and computes the performability.
- Assuming that it is possible to characterize the extent of service loss (possibly through the analysis of failure log) as different parts of the system fail, the performability value indicates the relative amount of useful service per unit time by the system in the

Graceful Degradation :- Performability Model(cntd.)



steady state and is proved to be equal to $Y = \sum_{i=0}^{N} \rho_i (1 - l_i)$ where ρ_i is the probability that the system is in degraded state i (i = 0 means no failure at all and no service loss, ie, $l_0 = 0$; $i = N$ means total failure with complete loss of service, ie, $l_N = 1$).

ERROR DETECTION

- Use of heartbeat and watchdog timers are traditional approaches to make the operational state visible to the outside world, which can help in detecting the failure of any node in a networked, real-time system.
- These techniques are certainly applicable in the context of an IoT system.

ERROR DETECTION – WATCHDOG

- This is a lightweight timer (often implemented by the hardware, as in traditional fault-tolerant VAX 11/780 or Pluribus reliable multiprocessor), which runs separately from the main process.
- If the main process does not periodically reset the timer before it expires, the process is assumed to have a control flow error (the correct flow of control would have reset the timer).
- In such a case, a hardware-implemented timer can generate an interrupt that can trigger a recovery routine.

ERROR DETECTION: – WATCHDOG(cntd.)

Therecovery procedure can:

1. Restart the process or the system from its checkpointed state
2. Invoke an appropriate recovery routine

- Popular open-source IoT middleware today such as RIOT or Contiki has a watchdog timer support.
- Snappy Ubuntu Core for IoT devices also reports that a watchdog mechanism is supported in the OS.
- Intel has recently announced their IoT infrastructure solution known as Intel IOT gateway, which promises hardware watchdog support.

ERROR DETECTION: – WATCHDOG(cntd.)

- Although a watchdog is easy to implement, the main problem with a watchdog-based approach is that it does not adhere to any specific fault-model.
- Therefore, it is not possible to figure out the reason why a watchdog-based trigger was activated.
- For a reasonably less complicated process running on an IoT device with a deterministic runtime, the watchdog is an effective mechanism for error detection.
- It is important to note that a watchdog is useful when availability is more important than correct functioning (reliability).
- Therefore, IoT scenarios where zero tolerance is expected, watchdog may not be a proper solution.

ERROR DETECTION – HEARTBEAT

- In this approach, a node in the network sends a message with a payload (ie, meaningful results, execution progress) to indicate that it is alive.
- If the heartbeat signal is not received within the prescribed time, the monitoring component assumes that the application running on the node has failed.
- Heartbeat based communication has been used over a couple of decades in distributed systems.
- In the IoT context, heartbeat-based detection can be employed when the interacting devices are computing complex time consuming tasks, or the devices are expected to respond only to some aperiodic events.

ERROR DETECTION – HEARTBEAT(cntd.)

- In such a case, it is important to know whether the device is fault-free and is ready to process any upcoming event.
- However, for devices that send data streams in a periodic manner, heartbeat- based technique can be an overhead.
- In such a case, watchdog-based approach is more appropriate.
- Identifying the correct time interval for a heartbeat is tricky.
- Many modern distributed systems use adaptive heartbeat mechanism where the heartbeat monitoring component estimates the heartbeat message roundtrip time (RTT).

ERROR DETECTION – HEARTBEAT(cntd.)

- A popular roundtrip estimation technique is $T = \omega T_p + (1 - \omega)D$ where T denotes the estimated round trip time, which is a factor of the previously computed round trip time T_p and the time-delay D to receive the acknowledgment after sending the heartbeat message.
- Following the architecture reference model, the heartbeat monitoring component can reside in the gateway module.
- For a distributed set of components without any central hub, the heartbeat monitoring needs to be implemented in the crucial components, who are responsible for data collection and processing from other nodes.

ERROR DETECTION: Exception Handling

- The basic premise behind an error detection technique is that an application satisfies a set of properties for all correct executions of the application.
- If the property is not satisfied for any run of the program due to a fault, exception occurs in the program.
- Use of exception handling mechanism in the program (using language level features, or through checking error codes) is a recommended guideline for a reliable software design.
- Such a design practice needs to be adopted in an IoT-based system as well.

ERROR DETECTION:–

Recovery Through Restart

- In conjunction with error detection, it is essential to implement a recovery mechanism for IoT components.

- At a minimum, autonomous devices as well as network devices need to have a basic restart mechanism, which can be triggered by the watchdog timer, or by the service platform.
- Restart is a useful technique for autonomous devices to recover from any transient error.
- However, it is often costly to have the entire system restart.
- Recently, a technique known as micro-reboot has been proposed where a module can be selectively restarted rather than the entire system.

ERROR DETECTION – **Recovery Through Restart** (Cntd.)

- Such a technique has been first put into practice for the J2EE system where an EJB container can be restarted rather than the entire application server.
- A component needs to be specially designed to have the micro-reboot
  - capability.
- In the context of an IoT system, the software written for an autonomous agent as well as the IoT service platform can be designed based on a principle called crash-only-design.
- Here the architecture of the system is not only based on loosely coupled components but micro-reboot enabled as well.

ERROR DETECTION – **Recovery Through Restart** (Cntd.)

- A micro-reboot capable component needs to maintain its own states in a state repository.
- Such a model has the functionality to create, read, and update the states.
- This functionality is invoked just before the component execution to load its most recent state.
- To implement the micro-reboot feature in the IoT service platform, the platform should adopt the micro-kernel architecture pattern such that the components dealing with failure prone external entities are decoupled from the platform core.
- In addition, the failure resilient driver model design can be adopted where the main service platform remains fault-resilient even when the modules that interact with various devices and other partner systems fail.

FAULT PREVENTION

- Fault prevention implies that faults are prevented from occurring on the runtime system.
- In traditional high availability systems, it is a common practice to detect an imminent failure possibility and remove a part of the system that can potentially fail.
- Although the basic idea is applicable in IoT, it is necessary to adapt these ideas in IoT-based systems.
- Another approach to prevent failures in high-availability system is to perform scheduled maintenance and timely software upgrades.
- Such a technique cannot always be applicable for the IoT devices.

FAILURE PREDICTION

- A generic approach for any failure prediction is to define a set of invariants the application must satisfy when it is in operation.
- If these invariants are broken, the application can encounter an imminent failure.

- If these invariants are modeled properly, it is possible to implement a monitor that can watch for a failure of any invariant and can take a preventive action.

The invariants can be designed at various levels:

- ➢ Program level
- ➢ Infrastructure level
- ➢ Process level

FAILURE PREDICTION – PROGRAM LEVEL

- Assertions in a program are invariants that need to be satisfied for a correct execution of a program.
- Program level assertions can be used for autonomous agents to detect an incorrect input at an early stage and prevent the error propagation.
- Program level invariants can be defined for the service platform software specifically at the places where the platform integrates with external subsystems, which are extremely vulnerable.

FAILURE PREDICTION – PROGRAM LEVEL (cntd.)

- There are automated control-flow-based assertion creations techniques that can detect:

i. if there is an unexpected branch from the middle of a basic block or

ii. it is branching to any illegal block.

- Such techniques use control flow analysis of the compiled program to insert the assertions.
- However, such an automated insertion of assertions can significantly impact the performance of the system.

FAILURE PREDICTION – Infrastructure Level

- For autonomous devices, one can define a set of system resource usage thresholds as invariants.
- An external monitor can observe if those thresholds are violated and triggers a preventive action.
- This is a common practice for a traditional HA system.

INFRASTRUCTURE LEVEL – PROCESS LEVEL

- Another class of noninvasive detection and enforcement of invariants has been proposed for a running system, specifically when the system is complex and it is not easy to insert assertions at design time or perform code analysis.
- These approaches do not deal with the source code but tries to collect invariant properties of the running processes from the operation logs.
- For such invariants to be meaningful, the challenge is that the invariants should not have many false-positive cases, ie, the invariant violations really do not indicate an erroneous behavior.

INFRASTRUCTURE LEVEL – PROCESS LEVEL (cntd.)

- The invariants should be at a reasonably high level so that the operations team can validate the invariant violations.
- These invariants, created from the past operational data, can be useful for the service platform.
- Specifically the flow invariants, which are relevant for detecting anomalies in the case of stream of data or a set of transactions, can be applied in the context of IoT service platform monitoring.
- A few flow invariant-based anomaly detection techniques for SaaS platforms based on statistical models have been indicated.
- A monitoring agent to monitor the infrastructure usage related invariants as well as use a stochastic failure model to predict an upcoming failure of a running SaaS system is proposed.
- Such an approach can be adapted in the context of an IoT system.

Improving Communication Reliability

- In many IoT application scenarios, the communication of machines and sensors without human intervention is a key requirement.
- To achieve this goal without failure, the reliability of the communication system that connects the sensors together plays a key role in the overall system reliability.
- Unlike traditional systems, various IoT application scenarios such as environmental monitoring and office automation demand that the communication infrastructure be reliable and sustain for a long time in an energy constrained environment.

Improving Communication Reliability (cntd.)

- For traditional system, use of redundant communication paths such as multihop networking has been a common design practice for better availability.
- Although such a communication protocol provides the needed redundancy in IoT, it is also important to optimize the number of hops, without sacrificing the reliability so that the energy consumption is under control.
- Optimizing the energy consumption in wireless network through optimal area coverage has been an important and active area of research for a long time.
- In particular, the energy conserving protocols such as UDP would be a better choice than TCP-based communication for the embedded devices.

Improving Communication Reliability (cntd.)

- However, UDP comes at a cost that it is not as reliable as TCP.
- To address the reliability problem, the Zigbee protocol (IEEE 802.15.4e) has been proposed recently which uses multihop communication to avoid a single point of failure and it is energy efficient.
- This has now been adopted in several IoT operating systems like RIOT, Contiki, TinyOS, and several others.
- A new reliable protocol for IoT where the broadcast nature of the wireless protocol is exploited to “overhear” a packet by the neighboring node even when the neighbor is not the intended recipient is proposed.

Improving Communication Reliability (cntd.)

- As a result, the necessity to send the additional ACK signal is alleviated.
- This in turn reduces the traffic in network, resulting in an overall improvement of the total energy consumption.
- Researchers have attempted to model various quality of service requirements of a wireless sensor network (which includes sensors, actuators, and the network)—namely the lifetime of a sensor, throughput, delay, and accuracy of data transmission which can be used to evaluate the reliability of a WSN.

SERVICE DEGRADATION SUPPORT

- **In the event of network failures, the network protocol may incorporate mechanisms to alert the autonomous objects participating in the interaction.**
- **Such objects can then trigger their in-built graceful degradation mechanism in response to the alert information received from the network.**

FAILURE PREVENTION BY SERVICE PLATFORM

- The service platform in the architecture reference model can help in making the IoT system failure resilient.
- The project MiLAN describes an environmental monitoring system comprising several embedded sensors and a central service platform.
- Although a sensor topology can introduce redundancy to reduce the single point of failure, the sensors may not have the overall sense of the topology to improve their longevity or to modify the next course of action based on the collective information generated out of the topology.

FAILURE PREVENTION BY SERVICE PLATFORM (cntd.)

- The service platform, equipped with the knowledge of the overall topology, can take a better decision to prevent any failure.
- In MiLAN, the middleware assumes the responsibility of improving the overall longevity of the sensor topology by regulating the quality of service of different sensor devices.
- The quality of the service for a sensor device is mapped to a reliability value to determine the state of the variable from the sensor data.
- For instance, in an IoT-based healthcare scenario a blood pressure sensor with QoS 0.9 indicates 90% reliability with which the sensor data can be used to determine the exact blood pressure.
- Depending on the overall state of the monitored patient, the middleware can switch off the blood-pressure sensor or can instruct the sensor to collect data at a QoS level of 0.9

IMPROVING ENERGY EFFICIENCY

- Since an IoT-based system is supposed to be operational for a long time, it is important that the autonomous devices can run on a battery power for a long time.
- To achieve this, a holistic approach to optimize the energy consumption needs to be considered to prevent any failure due to the unavailability of the battery power.

1. Device Power Management

- It is highly important to optimize the energy consumption by the device as the battery power can severely limit the device longevity.
- Dynamic management of power through dynamic voltage and frequency scaling (DVFS) is a common technique to reduce the power consumption of a device.
- However, it has been observed that scaling down the voltage can increase the rate of occurrence of transient faults on the embedded device.
- A recent task scheduling strategy considers a fault model related to DVFS and schedules tasks under varying voltage scales for real-time jobs.

1. Device Power Management (cntd.)

- CPUs in the mobile embedded devices are becoming increasingly powerful where multicore processors are being used in smartphones.
- In addition, these CPUs are equipped with various power management mechanisms such as offlining a CPU core or dynamic voltage and frequency scaling (DVFS) which reduces the CPU frequency to reduce the power consumption but compromises the performance.
- A policy that combines frequency scaling and core offlining can result in reduction of power without compromising the performability.

1. Device Power Management (cntd.)

- In this approach, the frequency of a core is increased to a threshold and then switches on another core, but drops the frequency to half for both the cores.
- This has been implemented as a new frequency governor in the Linux kernel, meant for embedded devices.
- Such techniques can be introduced in today’s modern sensor devices that are capable of running IoT operating systems for better energy management.

2. Communication Power Management

- **The communication protocols are becoming energy aware.**
- **Additionally, energy efficiency can also be improved by adjusting transmission power (to the minimal necessary level), and carefully applying algorithms and distributing computing techniques to design efficient routing protocols.**

3. SERVICE PLATFORM

- Researchers have focused on building the energy optimization capability at the IoT service platform level.

- Energy optimization by scheduling various activities judiciously where a selected set of nodes can be switched to sleep mode and only a subset of connected nodes remains active without compromising the quality of sensing and data gathering.
- The MiLAN project proposes an energy aware middleware to orchestrate the activities of the sensor networks.

3. SERVICE PLATFORM (cntd.)

- For example, the middleware can decide that even though the sensor topology contains redundancy, it may be more energy efficient to turn off a few redundant sensors so as to improve the overall longevity of the network under normal circumstances.
- Next, the middleware can reduce the overall load of the critical sensors in a particular application scenario so as to extend the battery life of the critical sensor.
- In yet another scenario, the middleware can reduce the quality of data (such as signal resolution) of healthcare devices in a normal situation.
- The quality of information (QoI) and energy efficiency in IoT network through a special purpose energy management module in the service platform, which can take an informed decision to switch on or off sensor nodes. The scheduler that is run by this module is aware of the QoI.

Data Quality Vs Energy Usage

- In the context of IoT, the data quality management has become all the more relevant specifically when the quality of data collected by the sensors has a direct impact on the energy consumption.
- The QoI present in the data broadly implies whether the data (image, environmental data stream, audio) are fit for using the intended purpose.
- Defining QoI metric is context specific.
- In general, factors like latency, accuracy, and other physical contexts (like coverage) can be used to create a QoI metric.
- An IoT middleware having a specific energy management module that provides an optimal covering of a set of sensor network devices without compromising the QoI using a greedy approach.
- Such an approach essentially aims to improve the overall reliability of the system and also increases the system longevity.