



Testing the **Internet of Things**

A GUIDE TO HELP YOU NAVIGATE THROUGH COMPLEXITIES OF TESTING
IOT SOLUTIONS IN TODAY'S HYPERSCALE CLOUD PLATFORMS.



Contents

Why Test IoT Solutions?	03
Overview of the IoT Ecosystem	05
Testing Areas within IoT software	06
IoT Testing Domains	08
IoT Testing Methodologies.....	09
The Big Picture of IoT Testing	11
Understanding the key IoT system test metrics	12
Functional Test Metrics	13
Performance Test Metrics	15
Network Test Metrics	17
Security Test Metrics	19
Device Modeling & System Simulation	22
IoT Test Automation & Artificial Intelligence	24
Conclusion	26
IOTIFY Network Simulator	27

Why test IoT solutions?

It is the year 2030, and the world has embraced singularity. All systems are automated and intelligent, thanks to the mass adoption of IoT along with Artificial Intelligence and Robotic Process Automation (RPA).

Imagine yourself as an engineer working for the maintenance of a state-of-the-art automated fire extinguishing system. It comprises of a few hundred thousand IoT sensors installed in dozens of building, and a bunch of edge and cloud-based servers, monitoring and analyzing data from the entire complex and its vicinity for any signs of fire incidents. This system detects & analyses the risk involved in any such incident in near real-time and triggers hundreds of RPA enabled drones to take off from the hidden canisters on the terrace. These super capable drones can then alight on the emergency exit windows on each floor and guide in evacuating people while spraying highly efficient chemicals to doze off fire in no time. You have just installed this system on a newly constructed hotel complex. All systems have been tested and certified by the manufacturers. Now it is time to do a mock drill to validate the end to end system response before the new system can go live. You initiate the mock drill and then wait for the system to spring into action.

Nothing happens!

You try again and initiate a false sensor warning for fire and the drones still do not crank up. The system is dead even before it is commissioned. The installation and commissioning teams get into a wild scramble only to realize that the system is choked after a power cycle and is unable to handle the scale of deployment of this new building due to a large number of DNS queries generated with so many parallel registration requests.

Whose fault was it anyway? Tester, developer or system integrators? What would have happened if there were a real fire incident and system wouldn't respond? Let's just come back to the present moment and think of the real complexity in designing and testing system such as these. IoT is a systemic congregation of man and machine. Machines are the force multipliers that augment our physical and cognitive abilities. However, this capability can turn into a disadvantage if certain aspects related to scale and security are not given their due validation.

The increase in the rate of adoption of IoT puts the spotlight squarely on existing testing practices and begs the obvious question. How could our testing practices evolve to match the speed of innovation in Software Development?



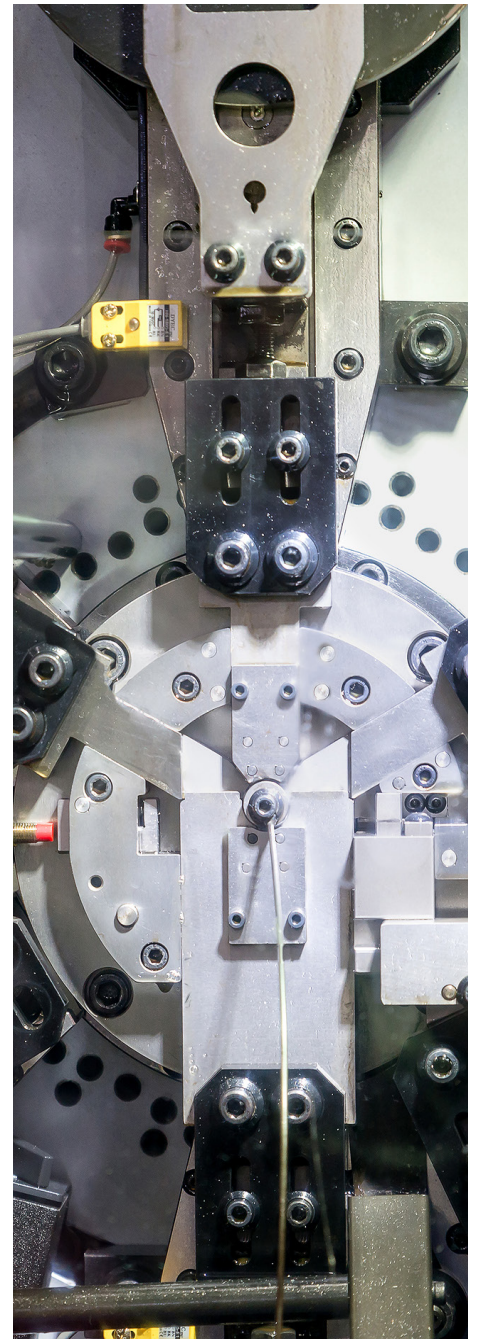
The complexity of the IoT system and scalability challenges demand a ground up and holistic approach to test the IoT solutions. The performance testing tools of tomorrow would need to evolve to match the complexity and scalability of the IoT. At IoTIFY, we were able to anticipate a clear need in the market to solve these challenges ahead in time and therefore we have designed our IoT simulation and validation software from the ground up to meet the needs of the future enterprise. We have used the same software infrastructure component which is used in top IoT platforms today to develop a performance testing software out of it.

While cloud-based software development would continue to evolve at a rapid pace due to the advancement of machine learning and AI, the testing practices are still playing a catch-up game. This eBook is a guide for all the testing professionals who are embarking upon this exciting journey of IoT system testing. An IoT system requires a different set of approach & mindset to design test strategies to bring about superior testing practices for the emerging technologies of the future. Through this eBook, you are going to embark upon this journey in following sections.

- Overview of the IoT Ecosystem
- Testing Areas within the IoT Software Stack
- IoT Testing Domains
- Understanding the Key IoT Testing Metrics
- Device Modelling & System Simulation
- Testing Tools Requirements
- Test Automation and AI

We hope you will enjoy reading through this ebook and gains some insights into how could you structure your testing approach to meet the stringent requirements of testing tomorrow's modern IoT hyperscalable cloud-based application.

We would welcome your feedback at hello@iotify.io



The important question to answer is How the existing test practices can be evolved to take on the complex challenges of information system evolution towards automation and intelligence, driven by IoT & AI?



Overview of the IoT Ecosystem

Internet of Things is getting mainstream adoption in the industry today. Estimation from leading Telecom service provider Ericsson says that around 29 billion connected devices are forecast by 2022, of which around 18 billion will be related to IoT.

Such an enormous impact of IoT is largely achieved by reducing cost, unlocking new business opportunities and enabling economies of scale. Software is going to play a major role in this ecosystem, and it is generally estimated that an average of 10% of the total cost of the IoT software will be spent on testing and validation of IoT devices and platforms.

The IoT ecosystem is also getting more and more complex. While the sensors and hardware are getting cheaper, IoT use cases are now crossing the conventional barriers of indoor confinement. Applications around smart cities, transportation, public safety, connected logistics are abound with IoT systems being deployed across geographical scale. With the increase in the geographical span brings in newer challenges in connectivity. A new set of WAN technologies have been developed for addressing the connectivity issues in portable IoT devices deployed in the field. This is being addressed under the ambit of LPWAN (Low Power Wide Area Network) protocols. Yet another complexity is the proliferation of different network topologies and communication protocols to address use cases which operate in mixed indoor/outdoor environment. Added to that the support for different software footprints for IoT devices, application stack, power requirements,

and we are looking at a very diversified set of protocol stacks which are not standardized. The war among these different protocols and topologies for IoT is akin to the battle of the preferred LAN protocol during the 1980s & 90s between two leading contenders of that time, the Ethernet and Token ring, with Ethernet eventually winning the game of mass adoption. In IoT, a dominating protocol is yet to see that kind of mass market adoption. However, it is racing very fast and the standardization of the software and protocol stack is going to be the game-changer.

Due to a multiprotocol-multivendor nature of IoT, in all probability, it will be much more complex due to the sheer diversity of human to machine (H2M), machine to machine (M2M) as well as machine to human (M2H) interaction and software is always at the center stage of this interaction. As the scale of IoT enabled interaction grows, the software complexity goes up and therefore the testing challenge increases exponentially due to the multifarious dimensions in testing and the effort in verifying the entire system. An end-to-end testing of IoT system presents a huge opportunity for the IT services and test automation industry which are the primary practitioners of software testing domain.

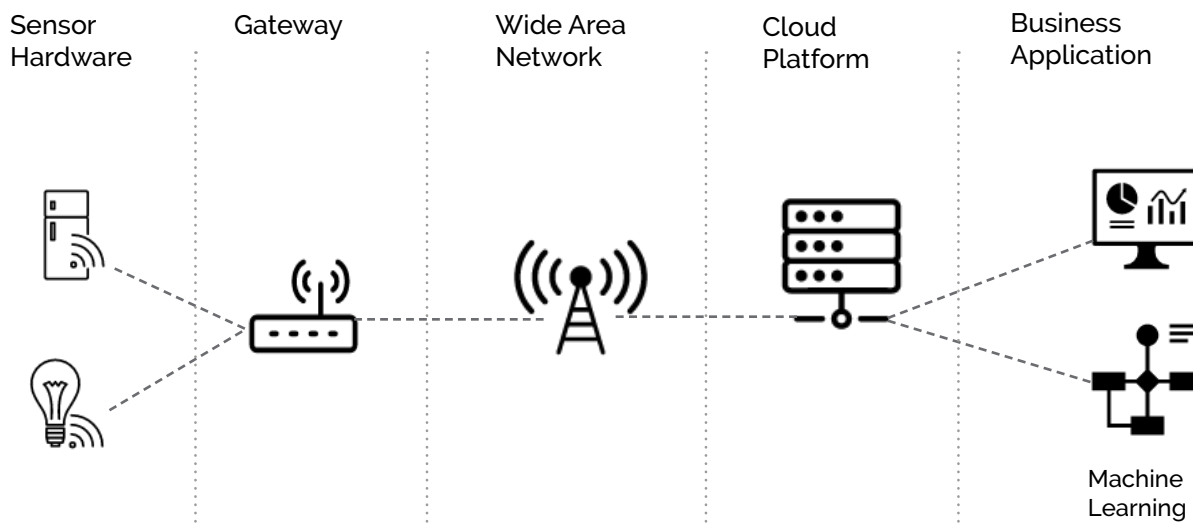
The IoT testing market is expected to grow to \$1378.5 Million by 2021 as per MarketsandMarkets (<https://www.marketsandmarkets.com/Market-Reports/iot-testing-market-51412648.html>)



The study from McKinsey group estimates that a total up to \$11.1 trillion a year in economic value could be generated by IoT by 2025.

Source: <https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world>

Testing Areas within IoT Software



To understand the role of testing in IoT, let's have a look at the overall IoT solution architecture. An IoT solution consists of many components and stacks which are depicted in the above figure with vertical line partitions. All these stacks have a software and hardware module that performs a specific functionality. E.g. The sensors in the hardware layer capture data which flows through the gateway WAN, up to the cloud server, where it gets processed and analyzed and then displayed in the user interface. Since each component handles the data in a unique way, the testing requirement for each layer is quite different. This is what defines the testing areas of IoT. Each testing area targets one layer within the entire IoT system architecture and addresses its functionality towards realizing the end-to-end IoT use case. Let's briefly cover each one of these testing areas.

A. Sensor Hardware Testing

The very first layer of IoT is the hardware which senses and interact with the physical environment. E.g. it could be a temperature sensor, a wearable heart-rate monitor. Hardware testing involves testing the hardware interfaces, peripherals, connectivity, power handling and other device functionalities. Due to its nature, the testing is done mostly manually and involves working with the hardware board with tools such as oscilloscopes, multimeters and signal generators. The software running on this hardware is called embedded firmware. From a software perspective, the focus is more on validating hardware-software interfaces, the accuracy of sensing and actuation mechanism and handling system failure scenarios such as malfunctioning flash or sensor data, software upgrade etc.

B. Gateway Testing

The gateway device sits on the edge and gathers data from multiple sensors and sends it to the cloud. The gateway testing requires validation of connectivity with the sensors and the business logic at the edge. Various IoT platforms provide docker based agents which can be deployed on edge devices with a centrally managed way. Scalability is

C. Network Simulation Testing

The objective of Network simulation testing is to understand the effect of network conditions on the connectivity protocol and test the resiliency of the cloud side software stack against network failure, packet loss and delays. It tests the cloud server functionality against real world situations such as a surge of traffic from gateways, DNS amplification attack, excessive packet losses, frequent disconnections and reconnection attempts. Testing on UDP based protocol is more interesting here as TCP has already a built-in mechanism to provide a reliable service. The testing requires sophisticated tools to model the network conditions and also the volume of traffic coming from hundreds of thousands of sensors.

D. Cloud Platform Testing

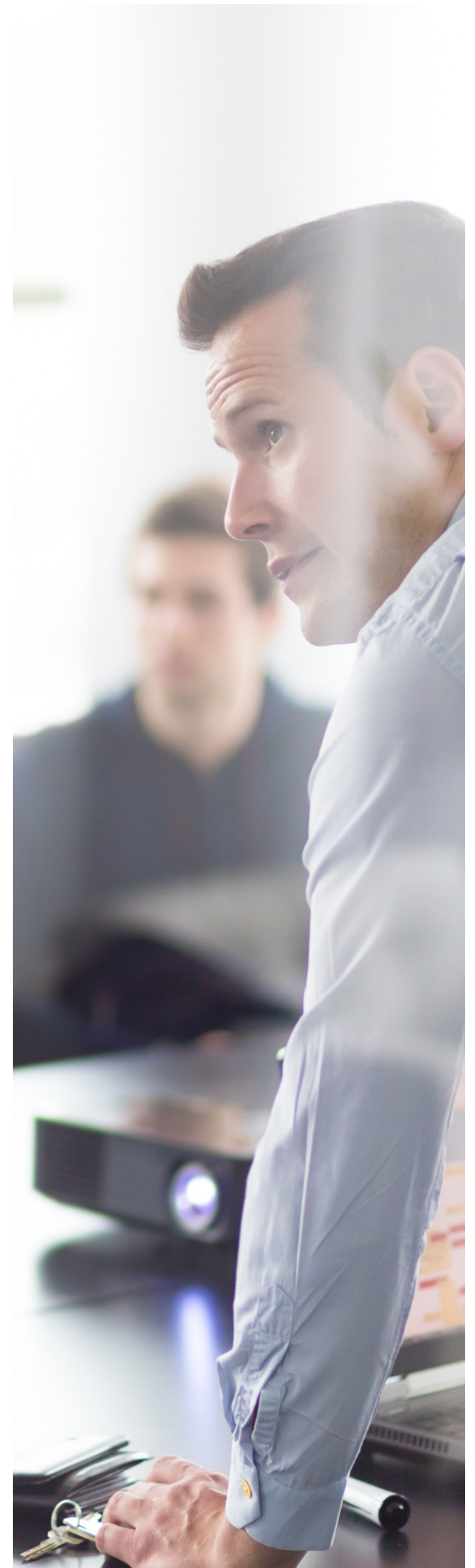
The cloud platforms are provided by service providers such as AWS, Azure, IBM and are responsible for ingesting and analyzing data at scale. The key criteria for cloud testing are functional correctness of application, along with testing for performance, analytics and scalability. One of the key challenges of testing the cloud applications is to understand the large set of inputs which can be generated by sensor and network conditions. The focus of this ebook will be primarily on this area of IoT testing.

E. Business Application Testing

Business applications are hosted on the cloud and are critical to how the user interacts with the system via the web, mobile and other external APIs. These are also responsible for presenting the data to users in a suitable form of visualization to make sure that the users get the correct information about system behavior. Testing of Mobile Apps and end-to-end user experience is also included in this.

F. Security Testing

Security testing is not an independent area but encompasses all other layers of IoT stack. At the hardware layer, security testing must ensure that there are no exposed open ports or telnet servers which are subjected to brute force attacks. Validating the secure boot, software download functionality and private key management for certificates is also critical. At the network and cloud layer, focus changes to ACLs, firewalls and providing role-based access to various system users.



IoT Testing Domains

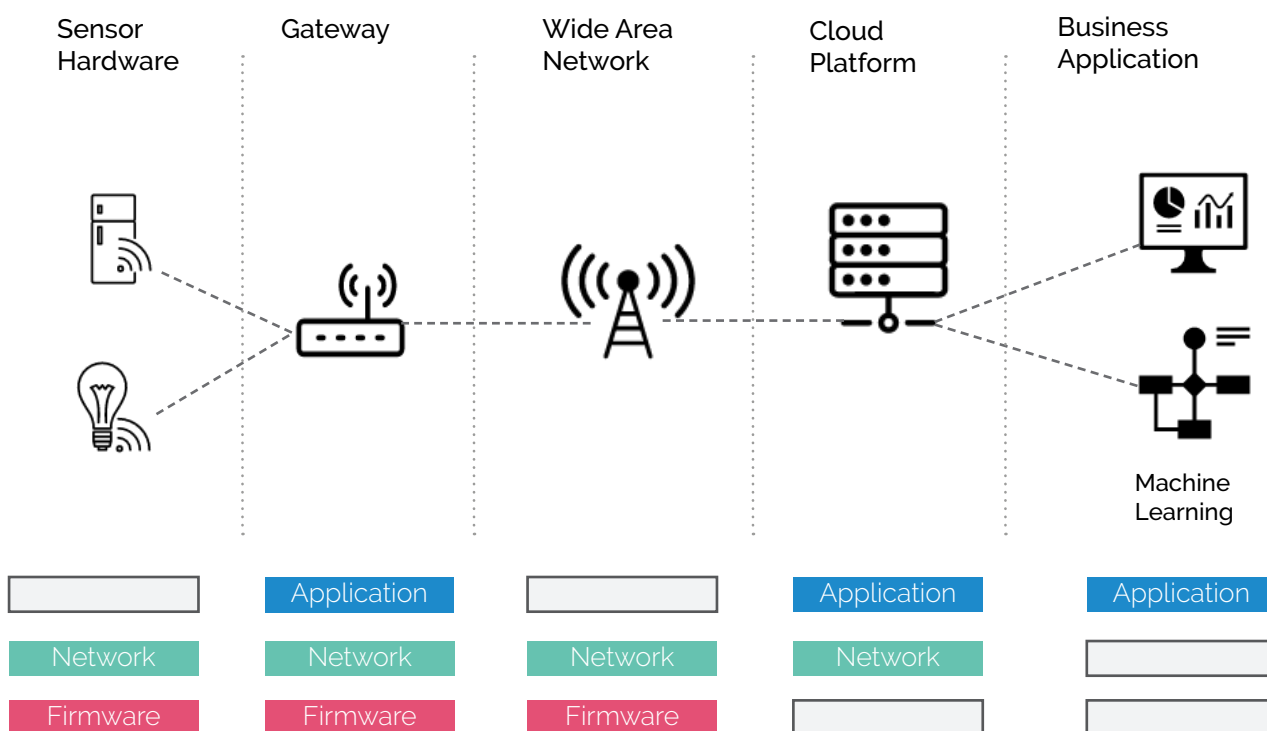
Now that we have covered the testing areas related to the software stack for IoT, it's time to outline domain expertise in testing. The testing teams could be organized and distributed based on their expertise in the the following three categories-

Application Domain: The team's expertise is in the cloud-based software module, which is responsible for the IoT application's business logic and cooperation with the application software running on the other IoT components. This domain of IoT testing requires skillsets of testers familiar with large enterprise applications, cloud analytics, real-time streaming, web and mobile interfaces.

Networking Domain: Connectivity software module which is responsible for physical connectivity and reliability of the data link. The domain requires expertise with TCP/IP, DNS, IoT Protocols such as MQTT, CoAP, HTTP, etc. The testing team is required to identify bottlenecks and issue with networking layers and making sure the system could easily ingest the data coming from millions of sensors.

Firmware Domain: The software which runs on the endpoint device or edge gateway, which is responsible for gathering sensor data and doing a preliminary analysis on it. This domain requires expertise in firmware testing, hardware-software interface testing, software update, etc.

Following diagram illustrates the relevance of the testing domain into IoT stacks.



IoT Testing Methodologies

Once we classify organizational role as per the testing domain, we can move towards classifying test methodologies.

Based on the IoT system component's operation and interfacing with other components, there are many types of testing methodologies that can be performed. Let's look at the important testing methodologies that outline the test strategy in undertaking the functional and experiential verification of an IoT system.

A. Functional Testing (FT)

The objective of functional testing is to validate the behavior of the component as per its defined specification. The specifications address the WHAT aspect, i.e. what is the component expected to do in response to an event. Along with that, the specifications also define certain constraints for the system operation.

All components, such as the sensors, gateway, and the cloud server define their specifications, while functional testing verifies that they adhere to their defined specifications and constraints.

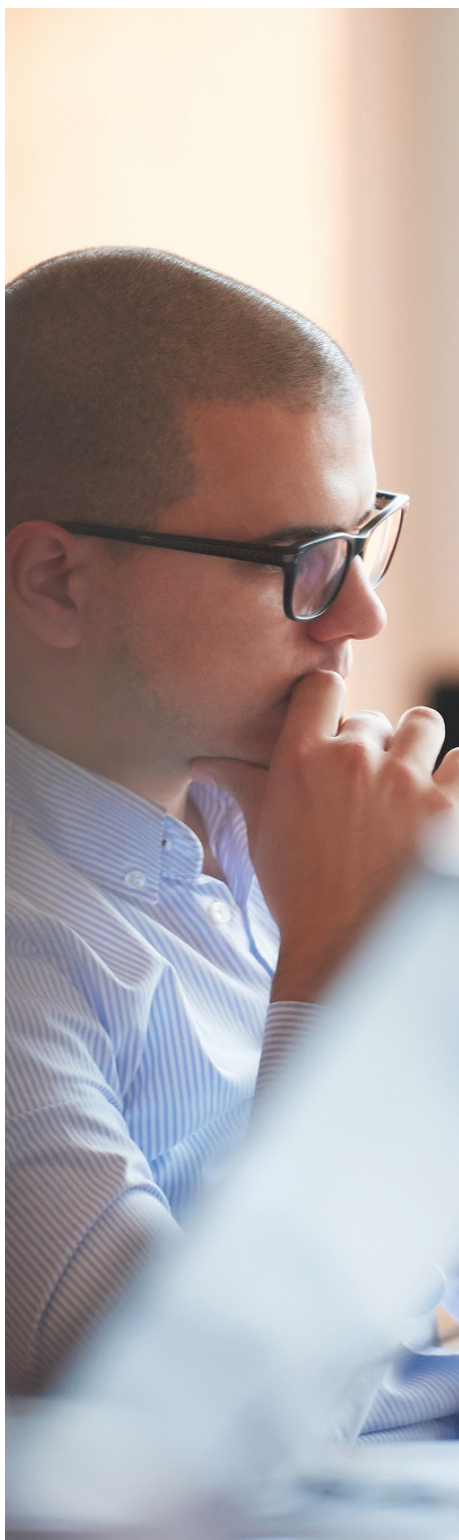
B. Performance Testing (PT)

Performance testing validates that the system can achieve its desired behavior within the given time constraints, especially when subjected to higher load conditions. Given that the end-user of the system is a human being, the components should lend themselves to behavior which must satisfy the usability constraints of the human user who is operating the system.

One of the key indicators for the need for performance testing is the performance requirement of the interfaces between two components. In the case of the IoT system architecture, the cloud server interfaces with many gateway devices to ingest the data from a large number of sensors. So, the cloud server must handle the data volume and process it fast enough such that the user interaction does not cause a lag. Handling data volume and producing responses within the given time constraint is just one of the aspects of performance testing. We will present some more metrics of performance testing in the following section.

C. Network Testing (NT)

Networking plays a very important role in an IoT system. There are Local Area networks which define sensor to gateway connectivity, such as Bluetooth, Zigbee, Powerline, etc. Then there are WAN networks which define gateway to cloud connectivity.



For that reason, many of the experiential aspects of the system are tied to the network's performance. Issues such as less bandwidth, erroneous data and delays in data transfer can hamper functionality and experience alike. The system should be able to not only perform under limited network connectivity, it should also be able to recover from frequent network outages reliably.

D. Security Testing (ST)


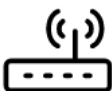



Security is a special case because it is not directly linked with functionality or experience, yet it can impact both in peculiar ways leading to what we call as "hacked" device. IoT security is a wide topic and is one of the most important issues amongst the decision-makers. Many of the IoT systems are mission-critical and a possibly of hack could jeopardize the entire business.

IoT security encompasses many areas such as secure boot, secure software access, data encryption, ACLs, firewalls, digital certificates, VPN and physical access protection and role-based access. At a bare minimum, security testing must ensure that the four basic tenets of secured communication are adhered to. These are

- 1. Authentication:** Any user or device trying to gain access to the system must authenticate himself as a valid user by means of a challenge.
- 2. Access Control:** Any user trying to read or write data within the system must have proper authorization to do so.
- 3. Data Integrity:** Any component receiving data from any other component must ensure that the other component is valid and the data is not tampered with or fabricated.
- 4. Encryption:** Any data generated, transferred, processed and stored by a component must be in the encrypted form such that only the sender and the intended receiver of the data can decrypt it.
- 5. Revocation:** System should validate that any authorized entity could be revoked access to the system and must not be able to access it past revocation.

The big picture of IoT Testing

Depending upon the component's functionality, its interconnection with other components, and its software stack, the need for a specific type of testing is defined.

	Functional Testing	Performance Testing	Network Testing	Security Testing	Remarks
Sensors 	Yes	No	No	Yes	1. In case of intelligent sensors integrated with SoC/SoM, functionality and performance testing becomes important. 2. In that case security testing is also important to make sure there are no direct ways to gain access to the component. 3. Physical security is always a concern for these components and needs to be addressed.
Gateway 	Yes	No	Yes	Yes	1. Performance testing will be applicable in case the gateway is a virtual device hosted on cloud with connections to thousands of sensors/actuators. 2. The gateways are the most vulnerable to cyber-attacks hence advanced security testing must be performed to plug all loop holes.
Wide Area Network 	No	Yes	Yes	Yes	1. The wide area network is not really an integral component of the IoT. Most often, it will be provided and managed by a third-party vendor, with some peripheral devices being part of the IoT system. It is important to make sure that this component does not open up loop holes to allow unauthorized access to the IoT system.
Cloud Platform 	Yes	Yes	No	Yes	1. This is the heart of any IoT system hence all types of testing are imperative and need to be performed at depth to unearth hidden flaws. 2. The connectivity layer does not play much part in this case as it is handled by the underlying network.
Business Application 	Yes	Yes	No	Yes	1. Most security breaches happen at this component as it is the direct interface for users. Deep security testing is imperative to eliminate any possibilities of unauthenticated or unauthorized access.



System Test Metrics for IoT

Functional Testing

Sample data set from each trashcan per event

Here is how the incoming events look like to the system

Trashcan	Timestamp (MM-DD-YY HH:MM:SS)	Latitude	Longitude	Fill level
3212	11-02-2018 13:24:52	40.73061	-73.95981	70
3214	11-02-2018 13:29:57	40.73051	-73.95983	30
5543	11-02-2018 13:34:59	40.72007	-73.94387	54
3236	11-02-2018 13:40:02	40.73010	-73.92015	55
4456	11-02-2018 13:44:10	40.77031	-73.34014	12

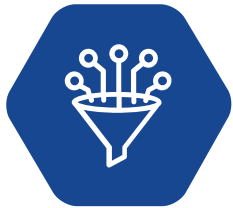
As a system tester, the following important points need to be understood about the system.

1. What is the frequency of receiving tracking events from each of the trashcan?
2. What data is received as a part of each tracking event?
3. What is the action taken by the system against each tracking event?
4. How does the system display the tracking event to the users?
5. What are the business rules for evaluating tracking events?

Tracking fill level of the smart trash cans is only one aspect of the functionality. The overall objective of the system is to increase the cleanliness of the city while improving trash collection efficiency. There are also some nondirect requirements here e.g. the lack of event: "What if the tracking event is not received since last 1 hour". This should be handled by the system by generating an auto-event which indicates action by the supervisor to manually check the trash can upon next pickup and either replace or reinstall it. All these functionalities must be captured in the specifications and should be properly understood to come up with a functional testing requirement metrics. E.g. the event processing system should be able to filter noise and spurious readings received from a sensor and show only correct values to the end users. It should be able to detect aggregate fill levels of an area given with a GPS coordinates and radius and classify those areas which should be urgently serviced before others. The system should generate an alert if no update is received in last 1 hour from a sensor. The system should be able to produce a visualization of the fill level of the trash can in Map based interface which is accessible from Mobile and Desktop. And finally, based on the real time fill level, system should be able to generate automatic collection routes in near real time for all pickup trucks as they start their daily collection trips.

Once the system requirements are understood, key functional test metrics can be derived as follows -

Key System Functional Test Metrics



Data Filtering

1

The system must be able to filter out inaccurate GPS, readings and corrupt data received from the sensors and only show a sanitized value to the end-user. The value shown in the display to the user must be smoothed (e.g. by calculating the average of the last 3 values received) so as to reduce frequent flickering in the visualization. Any incorrect value sent by a faulty sensor should be discarded and a manual inspection job should be scheduled if all values received from the sensor are being faulty for a long time.



Health Monitoring

2

The system must be able to individually monitor the health level of a sensor installed on the premises and generate an alert to the supervisor, if no data is received from the sensor for last 1 hour. The alarm should automatically be canceled in case Communication is reestablished. In case of low battery notification, the system should already schedule a service technician job to replace the battery of the sensor at the earliest.



Aggregation

3

The system must be able to show an aggregated visualization of the fill level in a map based visualization. By specifying a GPS location and a radius, the user must be able to see the location of all the trash can in the selected area and a heatmap of the current fill level. The visualization should also show the current location of the pickup trucks and subsequently show the result of trash can collections efficiency.



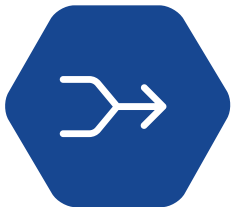
Route Optimization

4

The system must be able to generate a most efficient path for pickup trucks based on their carrying capacity, time of the day, crew duty hours and current status of trash cans. The algorithm should generate step by step navigation instructions for each vehicle to make sure they fill up the truck in the fastest possible manner and then reschedule to pickup from another required location, thus making the best use of their time, minimize the fuel cost and providing better sanity to the city environment.

System Test Metrics for IoT Performance Testing

Performance Metrics specify the efficiency and speed of the system to perform its functionality. They also specify the maximum load it can take and how the performance should be under peak load condition. The key performance metrics are continuous variables with a minimal and maximal range along with an average optimal range. For example, in Trashcan tracking application, the system must be able to handle a large number of multiple sensors sending their data simultaneously. It should also be able to filter out spurious or incorrect data. It should be able to run streaming analytics job on the incoming data streams to generate system alerts and automatically cancel them when condition is recovered or fixed. It should also be able to calculate most optimized routes for a trash can collection truck within a given latency. All these system requirements outline the key performance testing metrics a tester would try to baseline. Let's delve deeper into the topic of performance testing.



Ingestion per Second

1

The message processed per second is defined as the amount of messages fully processed and ingested within a given second on average, measured over an extended interval. Processing may refer to performing certain calculations, transforming the data from one form to another, storing the data or forwarding the data to another component. Messages coming to the system may take separate workflows, hence it is important to test the performance of the system under normal load conditions with a mix of event types.



Filtered per Second

2

The filtered messages per second is defined as the amount of messages ingested but discarded due to invalid data, or spurious readings within a given second on average, measured over an extended interval. It is important that filtration should happen as early as possible in the processing pipeline. The amount of computation spent on filtering message is also important because in ideal conditions it should be minimal.



Processing Latency

3

Processing Latency is measured as time taken from the system to fully ingest, analyze and display action corresponding to a single message on average. E.g. Let's say a trash can 100% full message arrived at the system at time T and the user was notified in the visualization at T+1 seconds. Then 1 second will be shown as the processing latency of the system. The latency will also differ based on the number of messages received in parallel, i.e. processing latency will be most likely increased when the number of messages received is higher per second.



Resource Usage

4

Measurement of CPU Usage, memory usage, database and network bandwidth utilization is critical to understanding the system health. It is also important to maximize the utilization of these parameters to ensure the cost-effectiveness of your solution, without compromising much on available buffer to handle the peak load conditions. E.g. an average memory utilization of 70-80% is optimum under normal working conditions.



System Uptime

5

Cloud servers are meant to be up 24/7 to handle ingress data from IoT devices as well as service requests from the users. The system should be continuously monitored to keep an uptime of 99.999% upwards. However, it is also important to handle the failures gracefully and recover successfully in case of a critical failure. Redundancy in compute, storage, database is critical to ensure a reliable system uptime. Thanks to the Microservice architecture, the system should be resilient to endure during a monkey test, where random disruptions are introduced in the data center (as if a monkey is running amok)



User Experience

6

Measuring the performance of user experience is also critical to ensure that there is no usability gap in the system when large number of users start using it. The time it takes for the UI to populate dashboard or fill maps with visualization is primarily a result of backend performance (such as DB Query) as well efficiency of browser front end. Using various UI automation tool, a limit should be established on page load and query times under stress conditions.



Graceful Degradation

7

The IoT system is designed to handle a peak load of input messages. What happens when it is subjected to a load more than its capacity? What happens when a key component is restarted and suddenly the system is subjected to a flood of messages?

It is important to analyze and measure what will be a graceful degradation of the system when subjected to abuse/malicious attack/unforeseen circumstances. It is also important to set a worst case performance metric which must be honored in such case and system is resilient to withstand such attacks.

System Test Metrics for IoT Network Testing

The area of IoT network testers encompasses the hardware, gateway, WAN and ingestion at the cloud. The tester should be familiar with the communication protocol being used (such as MQTT/CoAP) and messaging formats (JSON/XML) very well. Also the understanding of what each value means and what are the possible values for each range is critical to the good test coverage.

An IoT system is not complete without the interconnected components. There are diverse network topologies, protocols and connection options available for IoT, each suited for different use cases. Moreover, the network is always governed by an external network service provider. Hence it is a weak link in every IoT system and therefore it is crucial to address the IoT system response concerning network connectivity.



Bandwidth Utilization

1

Bandwidth is always a constrained resource in the network communication. It is important to find out the total bandwidth available in the system and then measure the current utilization of the bandwidth as a system parameter. Even though bandwidth is usually not a concern for the cloud system, it is important part when it comes to the cellular network or LoRaWAN technologies. In order to measure full system bandwidth a high performance message generator should flood the system closer to the ingestion layer. In cellular network it's not possible to ingest data from the network side, so a point at APN termination can be chosen (where the leased line of the carrier network terminates into your system). In case of DNS based load balancer is used, the maximum bandwidth measurement should be done with a geographically distributed load generator tool. Once the peak bandwidth is established, bandwidth utilization can be calculated.



Payload Size

2

All networks impose certain restrictions on the maximum size of the data packet, also known as MTU (Maximum Transmission Unit). It is important to test whether, under certain conditions, the IoT system generates a payload larger than the supported MTU of the underlying network.

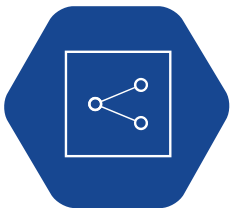
This issue needs to be addressed at an end-to-end level as the different components of the IoT system may utilize different types of network. For example, the GPS devices in smart trashcan tracking application will rely on a GPRS/EDGE network with less bandwidth whereas the cloud server and dashboard will be on broadband which has considerably more bandwidth. In this case, the tracking event payload sent from a GPS device should be within the MTU limits of the lowest supported MTU of the GPRS/EDGE network. Even if the payload exceeds the MTU, the payload will be segmented and reassembled at the cloud server end. This scenario must be tested to ensure that the system gracefully handles different payload sizes.



Packet Loss Recovery

3

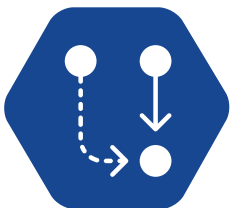
Packet loss is a very common phenomenon, mainly caused due to traffic congestion. For a wireless network, it also happens due to poor signal strength and manifests as either packet loss or erroneous packets. In almost all cases of TCP based communication, this is handled by the underlying transport layer of the network hence the IoT components do not have to handle it explicitly. However, if a UDP based protocol is used, it is imperative to verify that such situations should not impact their normal functionality.



Fan-In/Fan-Out Ratio

4

One of the most popular IoT protocol MQTT has a publish subscribe pattern, which means that any device can publish on any topic and any device could listen to any topic. One of the very interesting test case is to understand the fan in (large publishers, single subscriber) ration and fan-out ratio (Single publisher, large subscriber) in your system with an acceptable latency (e.g. 10 second). For example, if 1 message is published, how many subscribers could receive it within 10 seconds. This parameter gives us an idea of the network resiliency and availability of the infrastructure.



Switch over time

5

A network outage is a catastrophic failure which will render the IoT system dysfunctional. One way to get around this problem is to have a secondary network connection that acts as a backup and takes over in case of an outage in the primary network. However, this is not always possible due to environmental constraints. Whether the IoT system has access to the secondary network or not, either way, there needs to be a mechanism to handle the situation gracefully.

The load balancer and automatic failover system ensures that traffic coming from devices could be easily handled if one link breaks. The DNS based load balance would spread the incoming traffic geographically to the nearest node. The service health checkers would immediately shutdown a malfunctioning or corrupt node in case of an outage. From a system test perspective, the total service interruption time must be measured when such outage happens.

System Test Metrics for IoT Security Testing

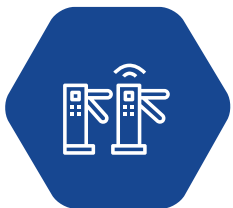
Based on the four tenets of security that we established earlier, we can derive the guidelines for security testing.



Authentication

1

Every IoT system must always support authenticated access to users and devices. The authentication mechanism is like the door lock of your house. It challenges you to take out the key to unlock it. Likewise, the authentication mechanism challenges you to take some action to prove your identity. If you consider the thousands of GPS devices being managed by the trash can tracking application, then the system must authenticate each device. The tester must ensure that the system rejects data from a non authenticated device while allowing it from a properly authorized device.



Role Based Access Control

2

It is the other most important feature of a system that guards access to sensitive information. In case of an IoT system, different users have access to the information at different levels and authorization mechanism ensures that only the authorized users can access the information. The trash can tracking application requires authorization for accessing various types of data such as the list of GPS devices, the list of users (collection vehicles) and the tracking events. These are general authorization roles. Moreover, there will be a special authorization for adding new GPS devices and users. This is akin to an administrative role. The test to verify support for authorization must consider all the possible ways to access data and must ensure that there is a role that guards access to each type of data.



Encryption

3

This needs no explanation as encryption ensures the privacy of data. Internet is a shared medium and most of the times we are dependent on an external network service provider. That makes it compulsory to encrypt the data end to end so that only the sender and receiver can decipher it. This also applies for data storage.

The use of encryption requires devices to store private keys and certificates. It is also required that these private keys must be stored safely in the hardware and can not be accessed if the device is compromised or hacked. A trusted platform module chip is important piece of hardware which ensures that functionality.



Data Integrity

4

Data Integrity means that the data can not be altered once its sent from the sender. Integrity protection ensures that there is a mechanism by which all components of the IoT system know and trust each other. If a component gets maliciously cloned, the other benign component should detect and report it to the administrator. There are various ways of achieving integrity protection based on shared knowledge, security keys and hash functions. The test for integrity protection must ensure that the integrity mechanism is verified and cannot be circumvented easily. The best approach is to have layered integrity protection that covers component to component communication as well as end to end communication.

Here are a bunch of additional things that need to be considered to safeguard every IoT system.

A. Handling DDoS

Distributed Denial of Service attack is a very potent attack being employed by hackers. Its purpose is not to gain access to the system but to incapacitate any computer that is connected to the Internet by bombarding it with legitimate-looking fake requests. Imagine if someone sends you a million junk emails on a given day when you were expecting an important email from a client. In the clutter of the junk emails piling up in your inbox, you will most likely miss that important mail. DDoS does something similar to a cloud server by keeping it busy handling junk requests while the legitimate requests keep waiting in the queue. It is important to safeguard the Internet facing cloud server from DDoS attacks by building a DMZ (De Militarized Zone) layer that can intelligently detect and debar all DDoS attacks from infecting the cloud server.

On the other side, it is also important to ensure that your hacked device does not become a part of botnet to perform DDoS attack. In September 2016, the authors of the Mirai malware launched a DDoS attack on the website of a well-known security expert. A week later they released the source code into the world, possibly in an attempt to hide the origins of that attack. This code was quickly replicated by other cybercriminals, and is believed to be behind the massive attack that brought down the domain registration services provider, Dyn, in October 2016.

B. Repudiation Management

One of the major aspects of security management is internal security. In the midst of handling security issues due to external factors, we can easily lose track of issue due to internal factors. Internal security hazards are caused by administrators, operators and other users, who operate upon the IoT system and can turn rogue to affect misconfiguration of the components of an IoT system. One of the obvious reasons for this occurs when the person is laid off from the job. Hence the user management and configuration console of the IoT system must implement a robust audit trail so that any change in the system can be traced back to the actual user who did the change.

C. Software Upgrade:

Lack of the software upgrade or patch is another major contributor to the security issues. A software upgrade procedure may open up additional gates (or ports) in the network interfaces of the gateway devices and cloud servers. It is important to ensure that these ports are managed and integrity checks are conducted with the remote server from where the software upgrade packages are transferred. Yet another security consideration for software upgrade is about the speed of rollback. If a new upgrade is found to have critical security issues then the upgrade procedure should be fast enough to roll back to the previous version. Moreover, the upgrade procedure itself should always stay benign such that it does not cannibalize itself. That is, a new software upgrade should not make the system dysfunctional such that any further upgrades or rollback become impossible. This can be achieved by keeping a last known good copy of the software in the persistent storage and revert back to it from the bootloader level when a success criterion is not established (such as no communication possible from the backend even after 1 day).

D. Diagnosis and Fault management

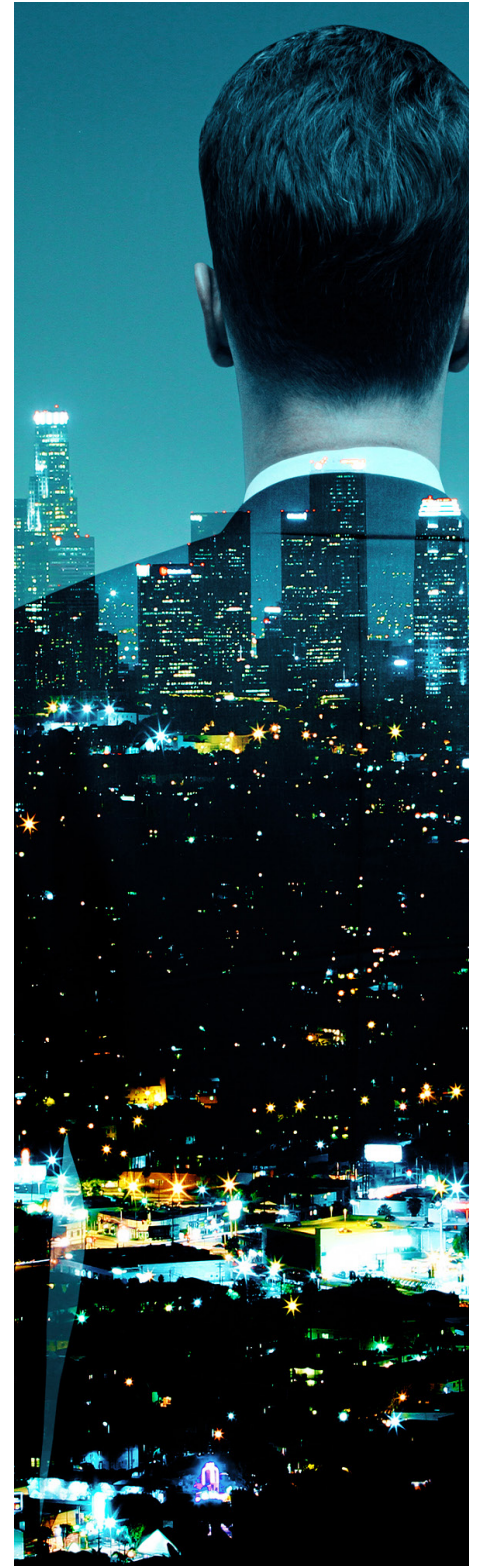
The IoT system must have a robust diagnostics and fault management mechanism which can aid the operators to take proactive actions in the event of suspicious activities such as

- a. Excessive requests
- b. Changes in network interfaces
- c. Abnormal CPU, memory or disk activity

It is imperative, therefore to have a good device logging framework and to automate the analysis of the logs by Machine learning. Any abnormal behavior such as constant SSH login requests, unauthorized software update requests etc. can then be flagged and marked for review.

E. Certificate Management

Enrollment, issuance and management of certificates is key to ensuring a robust and secure IoT network. There are various IoT specific protocols such as SCEP which can automate the enrollment of certificates at large scale. Vendors such as Azure IoT also provide device enrollment services.



Device Modeling and System Simulation

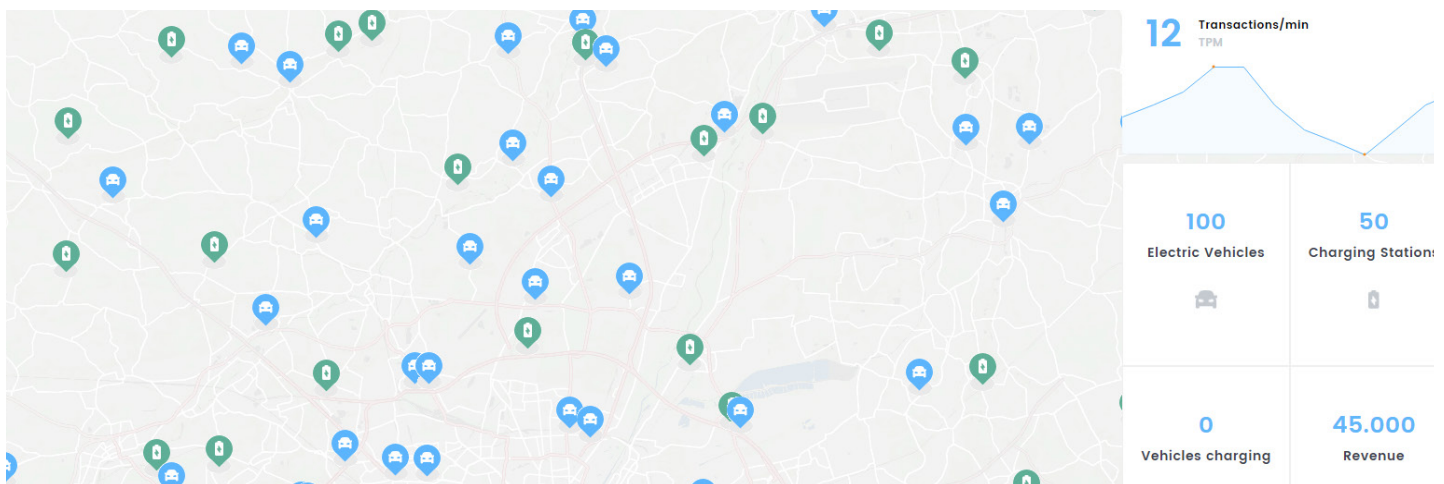
Traditional QA approach of pre-defining test cases and executing them sequentially becomes limited in meeting the exponential demands of IoT. The usual method of triggering a set of input and then validating the output from the system becomes suboptimal due to the nature of the system under test. Imagine that your test input vector needs to cover hundreds of thousands of corner cases. How would you achieve that with manual test case writing? A combination of varying sensor data, device states, combined with possible error scenarios, complex device to device interaction and unreliable network conditions will generate nearly an infinite amount of test vectors. A comprehensive IoT performance test needs to shift its focus on the macro level and define overall system test objectives. Simulation is the key solution to achieve an universal test coverage.

Modeling Device Behavior

The performance test objectives of IoT testing are completely different than traditional web testing. In the web domain, we usually monitor page load performance, API response time, the overall user experience and smooth user flow. Unlike the web, there are no humans involved in IoT devices. Instead, devices represent a combination of parameters known as "state". A state represents what IoT device is currently doing.

To take an example, when an IoT device is first installed, it will be commissioned or registered with the backend. Thereafter, it will mostly sleep and connect at regular interval to the backend. The IoT devices will monitor its various sensors at regular interval and then take a decision based upon a certain combination of sensor data. Depending upon the number of sensors, the device may do several actions and send several messages to the backend. Commands received from backend and message received from its peers would also influence the device behavior. Modeling such a complex interaction is not possible by recording a macro or any other method. Instead, this behavior needs to be programmatically modeled in your test.

Another aspect of building a test system is to model a device to device interaction. In the previous example of smart trash collection, whenever a truck goes to pick up a trash can, the fill level should be reset to zero. This needs to be modeled automatically in your test scripts.



A snapshot of IOTIFY's smart electric vehicle charging simulator



A scalable device model needs to take care of the following functionalities of devices -

- 1) Device Specific Context: A device has a unique set of identification parameter and run time variables, which are stored in memory/hard disk. The device may also have some attributed which needs to be persistent over time, such that location of a truck. The modelers should have a mechanism to store and restore the context of devices (Refer to glob concept in IOTIFY NSim to see an example of the system states can be stored)
- 2) Ability to interact with external environment: Once the test automation is running, you may want to trigger some manual actions which will cause a particular condition to occur in the system. E.g. if you are modeling a smart logistic scenario, you would want to introduce traffic congestion or closure in a certain section of the maps. Your device model should be amendable to handle such external requirements.
- 3) Ability to replay real-world data: One of the most useful feature of a device simulator is the ability to playback actual data captured from a physical device. The ability to slightly modify the replay would also add the benefit of analyzing the efficiency of the backend processes with much more accuracy.
- 4) Test analysis: Imagine finding a single error in the test logs of Million messages sent to the cloud platform. Imagine analyzing a single TCP connection failure while trying to publish messages. How would you go about debugging that? Your device modeling tool should give you the ability to analyze the results in depth.

System Simulation

Once the device models are perfected, its time to connect them together and create a system simulation. What is a system simulation? It's a full scale interaction of all the simulated objects sending data to the cloud platform as well as exchanging messages with each other. The goal of the system simulation is to let your devices play and interact with each other while testing how the cloud platform handles the control of the system. Let's take an example.

Imagine yourself simulating a smart city with hundreds of thousands of vehicle and CO2 sensor installed along the city traffic lights. When CO2 levels increase, the city lights will automatically redirect traffic away from the city and will delay more vehicles entering into the city by increasing the red light duration from the main entrances. This will cause user's navigation systems to take an alternative route expecting large amount of congestions within cities. How would you model the system and run simulation? Following will be the system requirements -

- Model several cars moving from random points between the city in real world traffic conditions.
- Simulate the interaction between moving cars and traffic lights.
- Correlate the number of passing cars from each junction to the CO2 level measured at that traffic light sensor.
- Report the GPS location of all cars and traffic light positions to the cloud platform along with CO2 readings.
- Test the algorithm running on the central server which changes traffic light duration and redirects traffic away from the city.

System simulation are usually customized applications tailer made for a specific purpose. [Contact us](#) to learn more about how could you build a system simulation specific to your use case.

IoT Test Automation & Artificial Intelligence

Despite the rapid advancements in software development tools and methodologies, software testing still hasn't evolved much. Use of scripting and test automation tools have helped testers a bit in offloading manual work, however the test planning, triaging and result analysis remain a primarily manual affair, requiring a lot of time and resources to execute. The situation becomes much complex in IoT. The traditional QA approach of pre-defining test cases and executing them sequentially becomes limited in meeting the exponential demands of certifying an IoT system as a whole. A combination of varying sensor data, device states, combined with possible error scenarios, complex device to device interaction and unreliable network conditions will generate nearly an infinite amount of test vectors. Thus, a comprehensive IoT performance test strategy needs to address all these possibilities at a macro level and define overall system test objectives.



Taming the Cyclomatic Complexity

Cyclomatic complexity is defined as the number of unique control paths in the software source code. The more the cyclomatic complexity, the more the number of inputs, their combinations and external conditions affecting the execution of software. For a complex software module with high cyclomatic complexity, the enormous possibilities of input combinations pose an unbearable cognitive load on the tester's mind. That is where AI comes to the rescue.

The use of Artificial Intelligence in software testing domain is touted to solve some of these problems. But how? AI is good at solving problems that are non-deterministic in nature. However, the goal of testing is to make the system behave in a deterministic way. Quite a contradiction. Hence the true fusion of AI with testing practice can only be achieved through a systematic shift in the traditional mindset of software testing process. Let's look at some possible approaches.

A. Test Case/Test Pre-Condition Synthesis

By feeding a combination of input data, AI models can be trained to generate test cases that cover all possibilities of input conditions. This way AI helps in exploring newer test vectors which can crop up due to new features and software releases.

B. System Behavior Analysis

For an IoT system, the end to end verification across all the components poses a humungous challenge for testing teams. This is because the combinatorial cyclomatic complexity across multiple components can lead to an unusual system behavior for a very rare combination of inputs and test pre-conditions. Defining test inputs combinations at such granular levels for multiple components poses an even more cognitive overload on the testers. Not only that, but it is also difficult to document such cases. AI can lend a helping hand by scaling up prediction and generation of the test data across multiple components. In this way, the end-to-end verification and acceptance testing of IoT system can be expedited to analyze the system behavior.



Large Scale System Simulation with Test Farms

Traditionally, the testing process entails running test cases and raising defects on those cases which fail to satisfy the expected outcome. However, this approach is not at all scalable for a multifarious system like IoT. This is where a test farm coupled with AI based test condition synthesis system comes to the rescue. A test farm is a large simulated environment for testing the system under test (SUT) subjected to various external conditions.

A. Exploratory Testing

An AI driven IoT test synthesis system can mimic human & device behavior, generate corner conditions and trigger rare combinations of test inputs and pre-conditions. All these individual combinations can be parallelly executed at scale within the test farm to produce test results for tens of thousands of test cases over a million iterations. This process can generate data that can be considered as Big Data. And this opens up opportunities to explore the system behavior in an even more granular and detailed level.

B. Predicting Non-Deterministic Behaviour

Irrespective of the complexity and scale, every system must be designed to exhibit deterministic behavior. That is a fundamental expectation. But in case of software, there are factors such as coding errors due to race conditions, deadlocks, multiprocessing as well as external factors that can make the system non-deterministic under certain circumstances. Through the multiple iterative testing on a test farm, AI can learn to analyze the system's expected output over a period of time, compare that with previous test runs and unearth the components and modules which pose the maximum risks in terms of non-deterministic performance. Amazon Web Services offers a service called Device farm, which is somewhat like this concept of test farm. It is used to run mobile apps across multiple types of devices and platforms (Android, iOS, Windows) along with triggers for varied interaction events.

C. Aligning Testing Practice with Business Intelligence

Businesses are evolving to a more AI driven approach to sales and marketing. Along the same lines, newer job roles have emerged to engage with customers and address the overall user experience of the products through experiments. In such changing times, the testing practice of a product must align with the business rather than merely taking inputs from system engineering teams, running test cases, raising defects and verifying them. This is where a leap in the traditional testing mindset is imperative. There are a few possible ways to champion this cause.

D. Test Synthesis Based on UX Analysis

One of the ways in which the testing practice can actively contribute to the QA is to start seeking data about how customers are using the product. Rather than themselves testing the products, the testing practice can build experiments to test user engagement, stickiness, overall experience and gather feedback. The data collected through these experiments can open up a goldmine of testing possibilities and provide some crucial insights on how users perceive the product. This data will also provide the testers with empirical evidence on usability statistics of the product such that this information can be fed back to the test synthesis system to build more robust test scenarios.

Conclusion

The Internet of Things is poised to fundamentally shift the way humans interact with their surroundings. The ability to fuse information from objects in the physical world automatically and analyzing that intelligently makes it possible to seamlessly merge AI with human activities. To ensure high reliability, availability, security and compliance of these autonomous systems is the biggest challenge for IoT system testers today. The depth and breadth of IoT application surface require multidisciplinary teams with diverse experience and at least a minimum common understanding of all the components of the system. The testing methodologies need to be optimized to enable highly efficient and automated testing of the systems and business processes, saving time for people and achieving faster time to market in product delivery and incremental functions.

It is also clear that traditional approaches for testing Web, Enterprise Software and Big Data do not apply directly to IoT systems, due to their deep stack technology, ranging from sensors and devices to cloud based analytics. A nearly infinite possibility of input vectors makes simulation a much suitable alternative to the system testing, where testers simply define system boundaries and let the simulation play across those boundaries. The multitude of technology also requires a set of advanced cloud based tools which are purpose-built to facilitate IoT testing.

By dividing IoT testing functions, domains and methodologies into neatly defined categories test teams could optimize and train resources to increase competency and deliver more productivity. The testing practice should also find out and define some key IoT system test metrics which describe the health of the entire system and serves as a benchmark for testing as well as development teams.

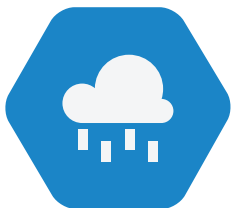
Finally with the help of AI, testing teams could automate mundane testing task and focus better on finding system limitations under load conditions and identifying the resource bottlenecks. The integration with CI/CD pipelines is essential to deliver a continuous performance and functional testing experience.



Introducing the IoTIFY Network Simulator

The complexity of IoT system and scalability challenges demand a ground up and holistic approach to test the IoT system performance. The performance testing tools of tomorrow need to evolve to match the complexity and scalability of the IoT. At IoTIFY, we saw a clear need in the market to solve these challenges ahead in time and therefore we designed our network simulation software from the ground up to meet the needs of the future enterprise. We have used the same software infrastructure component which is used in production today to build scalable cloud platforms and developed a performance simulation software out of it. The result is IoTIFY smart network simulator, which is first of its kind IoT performance testing software designed for cloud platforms. Here are the key facts which make it different.

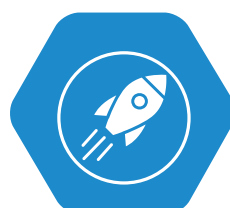
- A **cloud native** approach, which is horizontally scalable and could be run in any environment. The result is seamless scalability from Cloud, hybrid or event to the local desktops. The IoTIFY network simulator could truly match your IoT cloud platform and scale to test its true performance.
- **Advanced simulation** capabilities such as realistic traffic simulation, weather feeds, location functionality, and custom payload generation, with several helper libraries from a rich Node.js ecosystem.
- An **API driven** interface which can be easily integrated with existing test ecosystem and continuous Integration/Delivery pipelines. The test results could also be exported to any database of your choice.
- **Scripting** functionality in Javascript ES6, one of the most widely used scripting languages on the planet, we could tap into a rich ecosystem of NPM package ecosystem and could easily integrate Node.js packages into IoTIFY.
- **Multiprotocol support** – The scripting is protocol agnostic, which means testers could easily create several versions of the same test with different protocols. We support COAP, MQTT, HTTP, LWM2M, TCP/UDP out of the box and many other protocols will be supported in the future.
- Native Integration with **most popular IoT cloud platform** such as AWS IoT Core and Azure IoT Hub, enabling easy configuration and provision of certificates for million of devices with ease.
- Unmatched scalability of upto **1 Million device** endpoints or more. Ability to simulate large complex scenarios such as smart city, electric vehicle charging network etc. with customized dashboards.



Unmatched Scalability
upto 1 Million Endpoints



Fully Customizable Device
Modeling with Multiproto-
col Support



Once click deployment
& integration with
AWS, Azure



Full Test Automation
with CI/CD pipelines

Contact us today at hello@iotify.io to learn how could we help you automate your IoT testing.



Is your IoT solution **Ready to scale?**

Discover how our cloud-based IoT testing platform
could help you deliver scalable IoT Applications to
the market with full confidence.

iotify.io/contact



+ 41 -7988-18947



hello@iotify.io



Birchstrasse 21, 8212 Switzerland