MQTT—Enabling Edge-Device Connectivity in the IIoT Era

Abstract

Although the MQTT protocol has been around for nearly three decades, the design of the protocol makes it ideal for IIoT (Industrial Internet of Things) applications, the latest trend in automation engineering. This is particularly true for applications that stress "active notification," in which devices provide data only when needed, as opposed to "passive notification," in which devices are polled at regular intervals. MQTT's broker/client design eliminates the need for all devices in the system to be online at the same time. The clients (i.e., "devices" or "things") communicate directly with the broker, which plays the role of middleman to pass messages back and forth between clients.

Preface

Improving people's quality of life has always been one of the main motivations for seeking new and better technological improvements, and with the current push to connect more and more devices to the Internet, developing better products for so-called IoT applications is one of today's hottest topics. One of the biggest challenges for Industrial IoT (abbreviated IIoT) engineers is that the "things," often referred to as "edge devices," may not have ready access to a stable wire or wireless connection. Since the edge devices provide data (usually intermittently) to a central system, how to collect data from such devices is a big concern. Several protocols, including MQTT, AMQP, and CoAP, are possible candidates for meeting IIoT connection requirements. However, MQTT has become the top choice for most IIoT applications. In reference to Figure 1 below, more than half of IoT developers use MQTT as their communication protocol, providing strong evidence than MQTT is the best protocol for IoT applications.

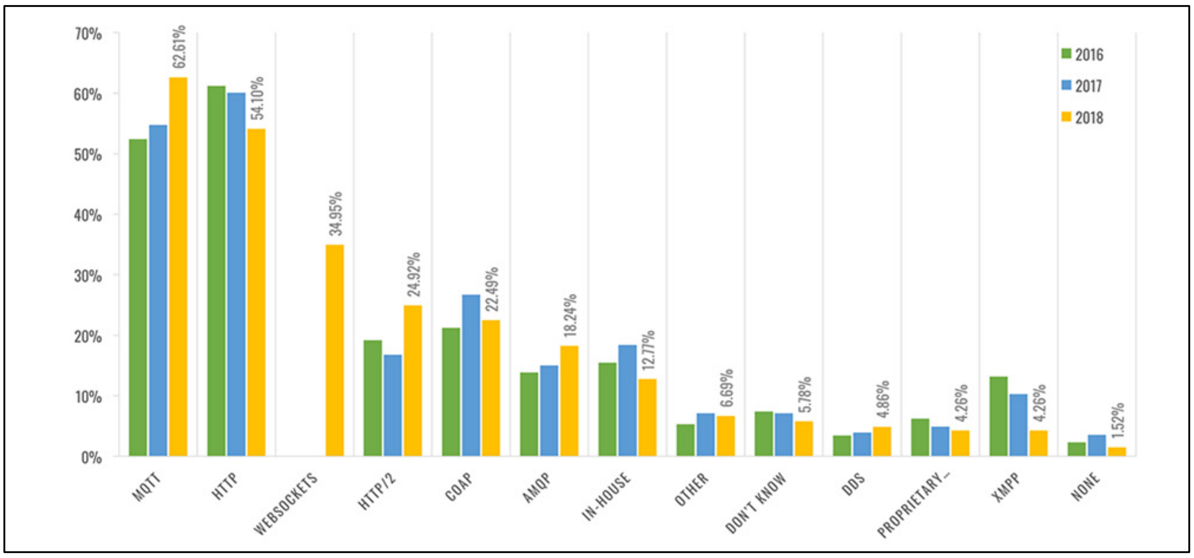

Figure 1. MQTT is the top protocol for IoT applications.

What Is MQTT?

The MQTT messaging protocol was first developed in 1999 by IBM and Cirrus Link, and was accepted as an ISO standard in 2013, starting with version 3.1. MQTT uses a publish-subscribe pattern (see Figure 2) to exchange messages. As illustrated in the figure, an MQTT system comprises one broker and several clients, where clients can either be publishers or subscribers. Publishers send data to the broker in the form of MQTT packets, which consist of a “topic” and “payload.” The broker then distributes the data to subscribers based on which topics they have expressed interest in.

The MQTT protocol does not specify a standard format for transmitting data, although it is common for applications to use either the JSON protocol or plain text. Compared with other protocols, MQTT has advantages that make it a perfect match for IoT applications.

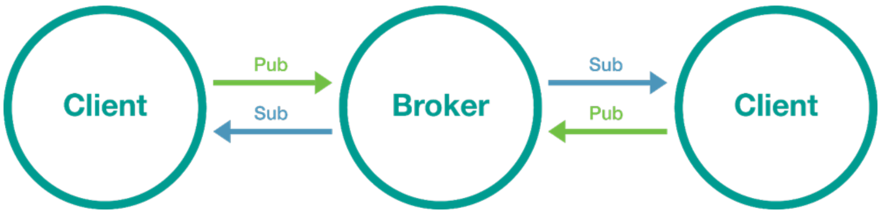

Figure 2. Publish-subscribe pattern

Publish-Subscribe Messaging Pattern

Compared with other request-response pattern protocols, the publish-subscribe pattern used by MQTT allows IoT developers to resolve certain common connection issues. For example, request-response patterns require both the client and server to be online at the same time to ensure that data is transmitted and received successfully. However, particularly for IIoT applications, it may be impossible for devices to maintain a strong enough connection to the network to receive the required data, and consequently, the request-response pattern is not suitable for such applications.

MQTT's publish-subscribe pattern is tailor-made for situations in which devices are not guaranteed to be connected to the network at the same time. The MQTT broker is crucial in this regard. The broker acts as an information center by accepting data sent to it from clients designated as "publishers" and then sending the data to clients designated as "subscribers." When the broker sends the data to a subscriber, it first checks to see whether or not the target client is online. If not, the broker can hold on to the data until the subscriber is online, and then send it. One advantage of this strategy is that only the broker needs to be online all the time. The clients—both publishers and subscribers—only need to get online when a connection is available, or when they need to send or receive data.

Event-Driven

When using a publish-subscribe pattern, MQTT clients only publish data to the broker when certain conditions are met (e.g., a warning signal could indicate that the temperature of a particular device is too high). Another way to describe this type of operation is that clients actively update data, instead of passively waiting for another device to request the data. For IoT applications, communication fees are charged based on how many data packets are transmitted. Compared with a request-response pattern, MQTT saves money since only one-way communication is needed to complete data transmissions.

Many-to-Many Communication

One of the main advantages of MQTT is that a publish-subscribe pattern can be used to easily establish many-to-many communication. The Machine-to-Machine (M2M) concept, which is one realization of many-to-many communication, is one of the hottest topics in IIoT. In factory M2M applications, machines at each station share their own process statuses with machines at other stations. Sharing information in this way serves to automate production optimization, without the need for manual input from operators. Since MQTT is used to implement M2M communication, machines only need to establish a connection with the broker instead of connecting directly to each other, saving a significant amount of time on handshaking. Since one broker is dedicated to handling the communication between all the machines, data transmission is more reliable.

QoS Design

The MQTT protocol uses three QoS levels to prioritize data:

- **QoS 0: at most once**
  In this case, the client publishes a message to the broker only once. The broker does not acknowledge receipt of the message or provide the client with any notification concerning communication with subscribers. The only guarantee is that the publisher knows that it has sent the message. However, it does not know if the broker or any subscribers have received the message. Although QoS 0 is by far the fastest quality of service policy, it is also the least reliable.

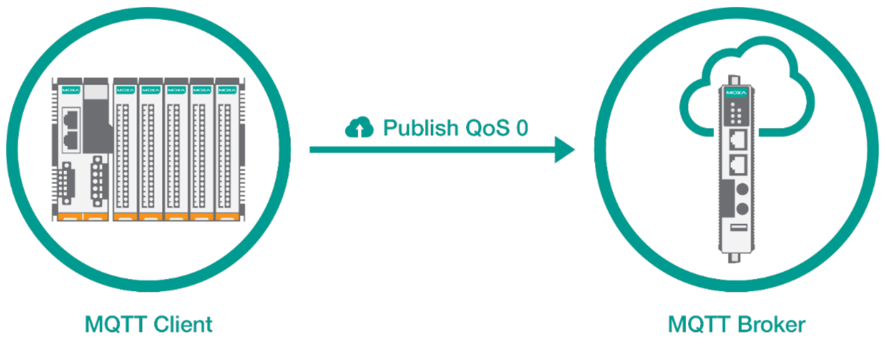

Figure 3. QoS 0: at most once

- **QoS 1: at least once**
  In this case, when a client publishes a message to the broker, the client expects the broker to acknowledge whether or not a client has received the message. If the publisher does not receive acknowledgement from the broker within a preset time interval, it will republish the message again and again until acknowledgment is received. Compared to QoS 0, QoS 1 is more reliable, although you can expect it to be slower over time.

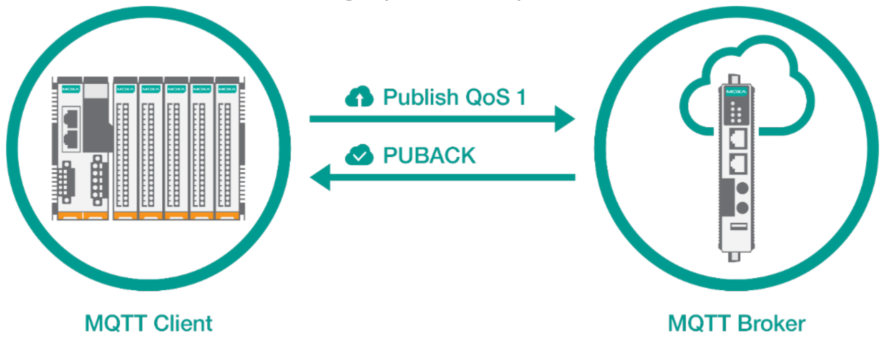

Figure 4. QoS 1: at least once

- **QoS 2: exactly once**
  In this case, the client and broker exchange four messages. The client first publishes the data to the broker, and then the client and broker exchange three messages: PUBREC, PUBREL, and PUBCOMP, to ensure that the data is delivered only once. QoS 2 is the most reliable, albeit slowest, MQTT quality of service policy.

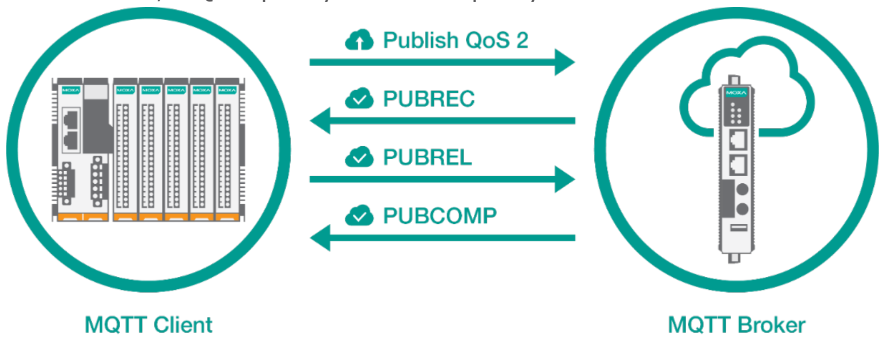

Figure 5. QoS 2: exactly once

Security

Security is a prime concern for IIoT applications. With more and more devices connected to the Internet, knowing how to minimize the chance of data getting hacked is a top priority. Figure 6 clearly indicates that security is by far the top concern for IoT applications. As far as

MQTT is concerned, the broker supports account names and passwords to prevent unauthorized clients from connecting to the broker to subscribe to topics. MQTT also supports TLS encryption for data transmissions to greatly minimize the chance that data will get hacked during transmission.

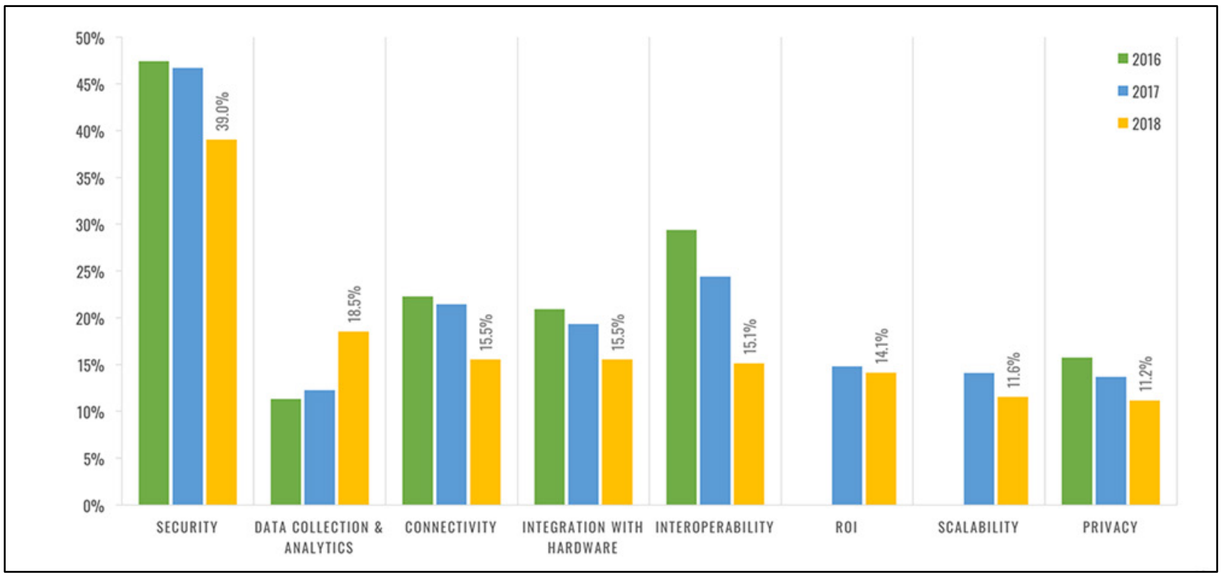

Figure 6. Security is the top concern when adopting the IoT

MQTT Application Architecture

As we pointed out at the beginning of this article, traditional OT applications are being refitted for IIoT applications that utilize the popular MQTT protocol. Two major system architectures are used.

Connecting Directly to the Cloud

Most public cloud services (AWS, Azure, Google Cloud, Alibaba Cloud, etc.) support the MQTT protocol to allow edge devices to connect directly to the cloud. To remain competitive and help shape the future of the industry, cloud services should at least provide the following benefits:

- Time savings

  Since the cloud service maintains the hardware (cloud server, CPU, memory, etc.), by leaving these more specialized maintenance tasks to the cloud service’s IT experts, users can spend more time developing their own solutions.

- Non-stop service

  Customers have come to expect near 100% reliability from cloud service providers, making reliability and network stability of prime importance. Every cloud service provider clearly states the level of guaranteed service stability they intend to provide in their Service Level Agreement (SLA). For example, in the Amazon Compute SLA, Amazon commits to 99.99%[1] monthly uptime, which means that service downtime can be expected to be less than 4.32 minutes per month. It's no stretch of the imagination to label this level of service as "non-stop service."

- Rich set of data mining tools

  Cloud services provide a rich set of tools, including data visualization, data algorithms, virtual machine, and machine learning, as part of their platforms. With access to such tools, users can more easily implement any number of diverse applications. For example, engineers can use a cloud service for data mining to reduce their server maintenance effort and to improve operational efficiency.

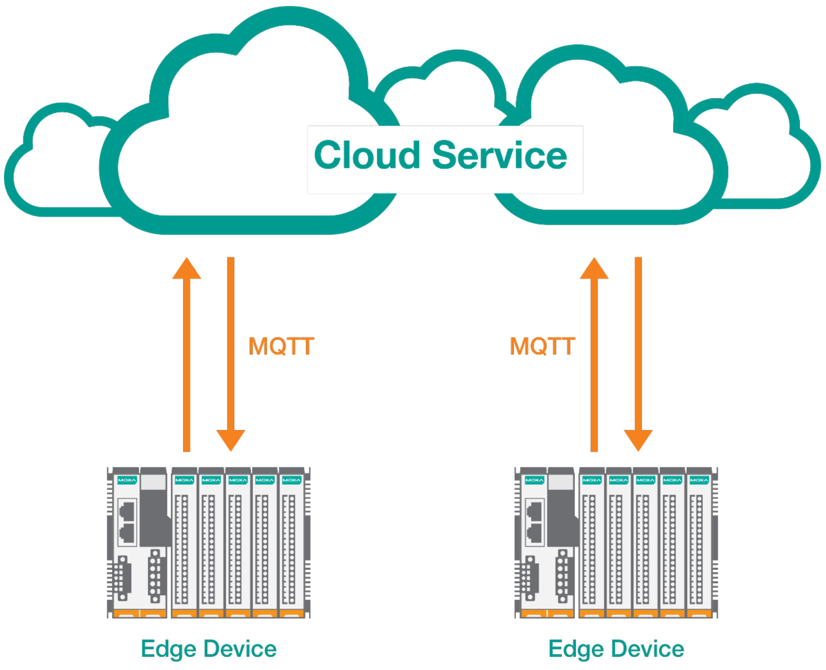

Figure 7. Architecture for connecting directly to the cloud

Connecting to a Local Gateway

Connecting edge devices directly to the cloud has benefits, but you should also be aware of various concerns related to adopting cloud services for IIoT applications.

- The first concern is cost. Since cloud services charge users by the number of data packets that are transmitted, it is not cost-effective to transmit data from edge devices to a cloud service directly. Even if the edge devices connect to the cloud via a cellular network, you still need to pay for the cellular service.

- The second concern is data security. Although cloud services provide well-protected environments for storing user data, some users are still hesitant to upload sensitive data to the cloud.

For most IIoT applications, setting up a gateway at the field site to collect edge device data and/or to enable M2M communication at the field site is one way to avoid these concerns. The gateway is usually an embedded computer, and although it does not necessarily need to be configured for both the MQTT broker and MQTT client roles, it could be configured as such. As an MQTT broker, the gateway can handle M2M data transmission at the field site. As an MQTT client, the gateway can collect field site device data and send the usable data to a SCADA system, HMI, or cloud service. The gateway solution further minimizes cost by using MQTT to enable M2M communication at the field site instead of through the cloud.

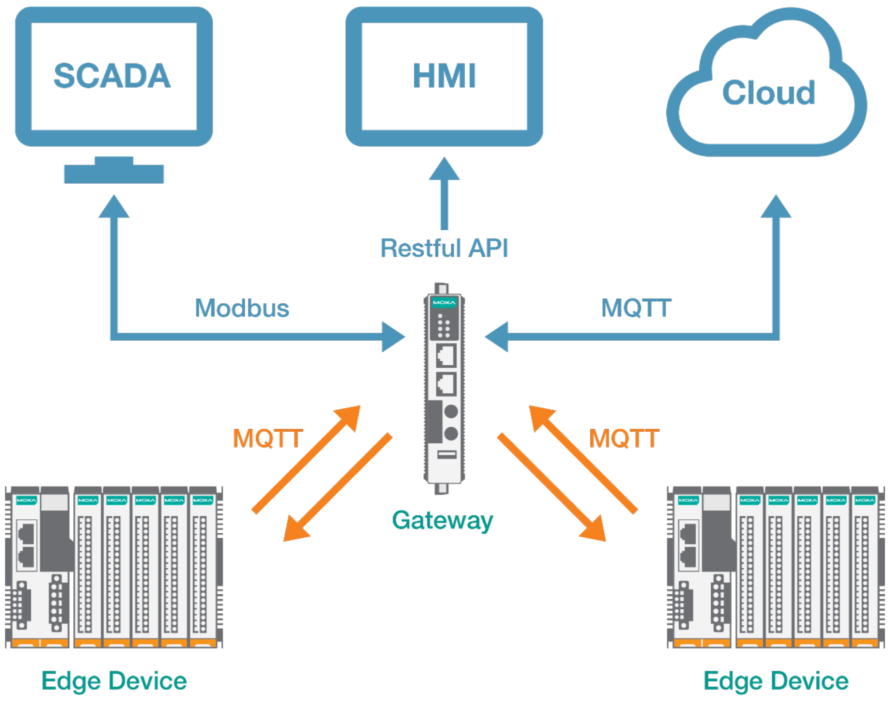

Figure 8. Architecture for connecting to a local gateway

The Challenges of Converting to an IIoT Application

You can expect to encounter some or all of the following challenges when transforming a traditional OT application to an IIoT application.

Legacy Devices Currently In Use Do Not Support MQTT

In factories, facility engineers generally use a remote I/O setup for data access and environmental monitoring. In addition, protocol gateways are used to collect power meter data and for monitoring energy consumption. With the IIoT trend now in full swing, if the MQTT protocol is going to be used to transmit data to the cloud, facility engineers will first need to survey and purchase new remote I/O products and gateways that support MQTT. With so many legacy devices still being used at field sites around the globe, converting a factory to an IIoT-based setup could require a huge investment.

Merging IT With Traditional Automation Applications Is Easier Said Than Done

One of the most basic aspects of an IIoT application involves collecting and transmitting OT data to the cloud, after which the data can be processed and/or analyzed. The challenge comes from that fact that the IT and OT industries use different transmission protocols. Modbus, which is one of the most widely used protocols in the OT field, uses data packets with small headers and payloads so that the packets can be transmitted over limited bandwidth networks. On the other hand, IT engineers use IT protocols, such as MQTT, RESTful API, and SNMP, to collect data, and consequently many IT engineers are not familiar with Modbus.

Security Is a Prime Concern

Maintaining network security is a prime concern for IIoT applications. From past experience, cyberattacks originate from outside the factory, so the first step to improving cybersecurity is to install a secure router, configure the firewall to keep the hackers out, and in general, upgrade your network security to prevent outside attacks. Edge devices on a factory intranet more often than not only support limited security functionality (if any), and still use unencrypted protocols. Modbus, for example, is commonly used to transmit data to and from edge devices. In recent years, certain high-profile cyberattacks put the spotlight on industrial network security issues. For example, in August 2018, TSMC was the victim of a cyberattack from a WannaCry variant, resulting in an estimated drop in revenue of about $200 million dollars[2]. The attack was a result of the fact that not all devices on the TSMC intranet had installed the latest security patch, which means that in principle, the attack should have been easy to prevent. The important lesson we can learn from this incident is that network security needs to be implemented, in some fashion, at the edge device level.

Moxa's Solution

Moxa's newly-released ioThinx 4510 Series modular remote I/O devices have key features that make them a perfect match for IIoT applications.

MQTT Client Support

The ioThinx 4510 Series supports MQTT client, which allows devices connected to the ioThinx 4510 to easily connect to cloud services. Although the ioThinx 4510 Series is promoted as an entry-level remote I/O product, its support for MQTT makes it a powerful asset. In fact, you can use the ioThinx 4510's user interface to define your own MQTT topics, and then based on these topics, determine which clients subscribe to which data. In addition to channel data, the ioThinx 4510 Series can also provide subscribers with data attributes, such as channel mode, maximum or minimum values, etc., so that IT-type devices connected to the network can get the latest status of an ioThinx 4510 product via MQTT. The MQTT payload uses the JSON format, which is widely used in today's IT industry. When subscribers receive an MQTT packet, they can easily search the data by a particular keyword "value" to find the data they are looking for in the payload.

Built-in Modbus Gateway

The ioThinx 4510 Series has a built-in 3-in-1 serial interface that can be used to implement a Modbus gateway. It only takes a few clicks to configure the ioThinx 4510 to collect data from a serial Modbus device. As with I/O data, the serial Modbus data is accessed by MQTT. Thanks to this function, both I/O and serial data can be collected using a single ioThinx 4510 Series device, greatly reducing the complexity and cost of your system. In addition to Modbus data, the ioThinx 4510 Series can access other protocols, including Modbus/TCP, RESTful API, and SNMP. In short, with the ioThinx 4510 Series, delivering serial data to the cloud is easy and straightforward.

Security Enhancements

To protect users' data, the ioThinx 4510 Series supports TLS v1.2 for encrypting data sent via MQTT transmissions. This widely used and recognized data encryption technology protects data transmitted over a network from third-party hackers. The ioThinx 4510 also supports account names and passwords for the broker to prevent data from being published to un-authorized brokers, and supports RESTful API via https and SNMPv3 so that all supported IT protocols can transmit with data encryption. For the Modbus/TCP protocol, which transmits data in plain text, the ioThinx 4510 Series has an access control function that relies on a list of IP addresses authorized to access the ioThinx 4510, thereby enhancing operational security.

With these advanced features and support for a variety of I/O modules, the ioThinx 4510 Series not only helps IT engineers collect field site data, it also helps OT engineers deliver the data to cloud services securely. In effect, the ioThinx 4510 Series eliminates the gap between the IT and OT worlds.

References

[1] Amazon Compute Service Level Agreement, Amazon Web Services from https://aws.amazon.com/compute/sla/
[2] Wu, D., Gurman, M. (2018, Aug. 5th). iPhone Chipmaker Races to Recover After Crippling Computer Virus, *Bloomberg* from https://www.bloomberg.com/news/articles/2018-08-05/iphone-chipmaker-races-to-recover-after-crippling-computer-virus