Osmotic Message-Oriented Middleware for the Internet of Things

Thomas Rausch
TU Wien

Schahram Dustdar
TU Wien

Rajiv Ranjan
Newcastle University

Editor: Rajiv Ranjan;
rranjans@gmail.com

Message-oriented middleware is a key technology in today's Internet of Things (IoT). Centralized message brokers facilitate decoupled device-to-device communication and can transparently scale to handle many millions of messages per second. However, Cloud-based solutions, such as AWS IoT or Azure IoT Hub, are challenged to satisfy the stringent Quality of Service (QoS) and privacy requirements of many modern IoT scenarios. Such scenarios are complex because they are not only distributed, but dynamic, as elements physically move, fail, and/or (dis-)connect to/from the network. Instead, distributed middleware needs to leverage the ever- increasing amount of resources at the edge of the network to provide reliable, ultra-low-latency, and privacy-aware message routing. But the heterogeneity and volatility inherent to Edge resources, and the unpredictability of mobile clients, make it extremely challenging to provide resilient coordination mechanisms and guaranteed message delivery. Applying Osmotic Computing principles to message-oriented middleware opens new opportunities for solving these challenges.

Message-oriented middleware (MOM) has undergone several architectural paradigm shifts over the past decades. The scalability issues of using centralized servers for application-layer message dissemination were discussed intensely during the peer-to-peer era, and many solutions based on completely decentralized peer-to-peer architectures were developed.[1] But the forces of monopolization and the widespread adoption of Cloud computing have caused many commercial solutions to return to a centralized approach, where scalability is largely achieved by consolidating resources into massive data centers. Amazon's AWS IoT,[2] or Microsoft's Azure IoT Hub,[3] for example, are massive transparently scaling centralized brokers that can handle billions of messages per second, even in the face of varying load, and at relatively low cost. However, despite

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

further advances in Cloud computing and clustering techniques, the IoT and related application scenarios have introduced new challenges, in particular for centralized systems. IoT applications with ultra-low latency, network resilience, or stringent data privacy requirements cannot be implemented by Cloud-based solutions alone, but demand decentralization. Yet, system engineers have grown accustomed to the convenience and benefits of centralized deployment and management.

Edge computing has been recognized as a solution to this dilemma.[4] By leveraging the ever-increasing amount of resources at the edge of the network for computation, data management, and application-layer message dissemination, many of the QoS and privacy challenges can be addressed. Edge computing also recognizes the Cloud as a necessary and integral part of a holistic system. The Edge can hand off compute intensive tasks to the Cloud, or forward processed data for long-term storage and offline analytics. But how to efficiently and effectively integrate the Cloud and the Edge remains a topic of debate. The heterogeneous nature of edge resources, and their unpredictable availability make it challenging to provision and manage them in a centralized manner. Client mobility further complicates matters, especially with respect to message delivery guarantees and QoS optimization in MOM.

Osmotic computing has been proposed as a new computing paradigm for Edge/Cloud integration.[5] It borrows from the biological principles of osmosis, where solvent molecules seamlessly diffuse into regions of higher solute concentration. Although osmotic computing was originally conceived for the dynamic management of microservices across federated Edge/Cloud infrastructure, it has enjoyed success in other problem areas such as stream processing for IoT or deep learning.[6–7] We argue that osmotic computing principles can be applied to seamlessly integrate Cloud-based MOM into Edge computing. Osmotic computing can abstract how we think about elasticity of MOM towards the Edge, i.e., dynamically moving or provisioning message brokers from the Cloud the Edge based on current demands. In particular, we propose an architecture based on two diffusion models: broker and client diffusion. From a static centralized deployment in the Cloud, we bootstrap a network of brokers that diffuse into the Edge based on resource availability, and the number of clients and their proximity to Edge resources. Clients in close proximity diffuse to the Edge depending on broker proximity and the potential to optimize QoS between the clients. We present an architecture, and proximity-detection and orchestration mechanisms to implement MOM based on osmotic computing principles.

THE NEED FOR EDGE-ENABLED MOM

Modern IoT scenarios highlight the need for MOM that leverages resources at the edge. For example, let us consider the following scenario from the mobile health (mHealth) domain. In a major disaster situation, effective coordination and prompt reaction of emergency teams is paramount to save people's lives. To support on-premises decision-making, first responders attach wearable biosensors to patients, which continuously publish medical signs. Emergency technicians carry mobile devices with software to visualize patient data, trigger alerts, and coordinate team efforts. Cloud-based IoT platforms may be impractical or unreliable in this case, as such health-critical mobile systems must be resilient towards network partitions, and require near-real time data processing and message exchange. Instead, resources in close proximity, such as tactical cloudlets[8] deployed in emergency vehicles must be used to provide necessary services. From there, data can be processed, forwarded to hospitals for further analytics and acute patient care, and handed off to the cloud for long-term storage.

MOM plays a critical role in this type of scenario. It has to facilitate low-latency communication between devices in close proximity, e.g., to provide immediate alerts to changing medical conditions, while still being able to disseminate messages to locations on different levels of geographic dispersion. However, the mobility and unpredictability of both clients and resources that could act as brokers require a completely dynamic operational mechanism. In Cloud-based MOM, operational configuration, such as broker addresses, publish/subscribe topics, etc. are typically best defined at deployment time. In contrast, Edge-based MOM needs to react to spontaneously emerging network structures and device congregations, such as our disaster scenario. This

also means that clients need to be able to quickly locate the closest broker, and QoS optimizations need to consider that proximity between clients and brokers may change during runtime, leading to dynamic routing as end and edge devices change location.

State-of-the-Art

Messaging middleware, publish/subscribe systems in particular, have enjoyed immense research interest over the past decades. Systems like Hermes[9] or PADRES[10] have led to an entire body of research dedicated to optimizing and enabling robust message dissemination in complex peer-to-peer overlay networks. The advent of Cloud computing and the IoT has changed the landscape of real-world messaging solutions dramatically.

Over the past few years, many commercial cloud-based messaging solutions have emerged that target IoT platforms. Amazon's AWS IoT[2], Microsoft's Azure IoT Hub[3] and others all aim at providing transparently scaling IoT device integration via pub/sub messaging. Although these solutions can deal with massive amounts of devices and messages, they cannot satisfy the stringent QoS and latency requirements imposed by many IoT scenarios, as they completely disregard proximity and edge resources.

Some open-source solutions have shown tentative efforts to provide basic mechanisms for enabling real-world edge computing applications. The popular message brokers Mosquitto or HiveMQ, for example, provide the concept of bridging, where brokers can be deployed at the edge and statically configured to forward messages of specific topics to centralized brokers.[11–12] JoramMQ[13] can model a hierarchical broker tree to allow low-latency communication in pre-defined subsystems. These approaches are extremely limited in their operational capability, as they are all static in nature.

Only recently have researchers started to investigate the challenges of IoT and proximity awareness that confronts messaging middleware. MultiPub,[14] for example, manages a broker cluster that spans multiple cloud regions, and optimizes latency for clients based on their proximity to a region. EMMA[15] goes one step further in that it considers brokers on arbitrary edge resources, and dynamically re-configures client-broker connections at runtime. PubSubCoord[16] has also identified Edge/Cloud integration as a challenge for IoT, and proposes a dynamic broker system consisting of edge and routing brokers.

OSMOTIC MESSAGE-ORIENTED MIDDLEWARE

Design Goals of Osmotic MOM

As we have illustrated, the IoT and edge computing architectures provide a great deal of challenges when building messaging middleware. Based on our motivating scenario and existing analyses of edge computing characteristics,[17] we have identified several key design goals that drive the design of our architecture and control mechanisms.

Stringent Latency Requirements

Optimizing end-to-end latencies for clients is one of the key goals of our middleware. Most of the latency in messaging middleware is caused by packet routing, lost packet retransmission processing, and physical propagation delays of network links. Routing messages to the Cloud and then disseminating them from there is wasteful, especially if device-to-device communication happens in close proximity. For these situations, brokers deployed at the edge can provide ultra-low latency communication. At the same time, subscribers may be located at geographically dispersed locations, e.g., a data analytics services in the Cloud, meaning that messages must be forwarded to the Cloud if necessary.

Mobile Brokers and Endpoints

Mobility is a fundamental element of IoT[19] and Edge computing and therefore must be considered as such by osmotic middleware by design. In our scenario, not only clients, but also brokers can be mobile. Proximity between nodes may change any time during runtime, and the system should be equipped with mechanisms to maintain optimized QoS, even in the face of client or broker mobility. Providing message delivery guarantees in the face of mobility is another prominent challenge.

Variable Resource Availability

Resources at the edge are a key component in enabling the previous two goals. However, because these resources are highly dynamic, we cannot predict or rely on their availability. We therefore also cannot provide static operational configurations. Instead, a central configured deployment should be able to organically diffuse into edge regions where and when necessary. A key mechanism required to facilitate this is proximity detection.

Monitoring and Management

Looking at the previous goals, it becomes clear that proximity is a fundamental concept in IoT and edge computing, and needs to become a first-class-citizen not only in messaging middleware, but edge computing algorithms and systems in general. This would appear to imply that location of brokers and endpoints must be determined in real-time and their physical relationship to each other must be deduced in terms of the local topology. As we'll see, proximity can instead be determined by calculating effective network latency. Additionally, routing of messages between elements must balance reliability and performance considerations. Moreover, scalability of proximity monitoring is a key challenge to avoid excessive strain on the network and resource constrained devices.

IoT Heterogeneity

A plethora of messaging protocols and client hardware for the IoT exists that may be difficult to replace or update.[18] Existing infrastructure should seamlessly integrate with a distributed messaging middleware. Furthermore, middleware control mechanisms need to be completely transparent to end devices.

Architecture

Figure 1 shows a high-level overview of our architecture, which includes the components that follow.

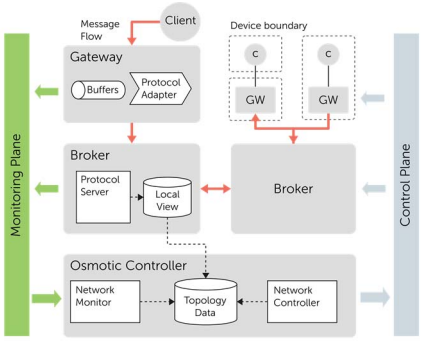

Figure 1. The architectural components of osmotic message-oriented middleware. The osmotic controller orchestrates a network of brokers and gateways via a monitoring and control plane. The gateways transparently connect clients to the system.

Broker Network

Brokers implement a server-side messaging protocol (e.g., MQTT), and facilitate message dissemination to clients and within the broker network. Because brokers are treated as an ephemeral resource, their control mechanisms have to be entirely dynamic. This particularly concerns addressing and message routing. However, maintaining a global view of the network topology at each broker would be impractical. Instead, the Cloud maintains a global view, whereas brokers only maintain a local view of the topology and routing tables necessary to operate within their contextual boundary. Brokers also monitor and provide access to key performance indicators, such as resource consumption, message throughput, average routing delay, etc., to provide coordination and load balancing algorithms with necessary data.

Gateway Network

Gateways are an integral part of the coordination fabric and provide clients transparent access to the broker network. They serve two main purposes: 1) to seamlessly integrate existing IoT infrastructure into the network, and 2) to enable client and broker mobility. Gateways act as intermediaries that intercept protocol control packets from clients to understand their context (e.g., connection information, Message Delivery Agreements (MDAs) and topic subscriptions) and allow the transparent reconnection of clients to other brokers when adapting to varying network QoS or topology changes. They serve as proxies to measure proximity between clients and brokers. Gateways also do simple data processing such as filtering or protocol translation via protocol adapters. Because they are lightweight, gateways can typically be easily hosted by resource-constrained IoT devices, and can serve either one or multiple clients.

Osmotic Controller

The controller maintains a full view of the network topology and is responsible for bootstrapping and orchestrating the system. New brokers and gateways register with the controller when they enter the network, and continuously report data to the controller via the monitoring plane. The controller acts as a registry and discovery service for edge resources to which brokers could potentially be provisioned. Brokers are not dependent on the controller to process and forward messages because they maintain a local view of the network topology and routing tables. However, the osmotic controller is required to 1) diffuse clients, i.e., enacting QoS optimization decisions by instructing gateways to connect to other brokers, and 2) diffusing brokers, i.e., elastic provisioning of brokers to edge resources.

Protocol Adapters

Protocol adapters are a subcomponent of gateways and a fundamental part in controlling the heterogeneity inherent to IoT. With these adapters, gateways can act as protocol bridges. For example, clients may send messages via a different protocol than implemented by the middleware. To seamlessly integrate these clients into the system, a protocol adapter on the gateway translates messages from the client protocol to the server protocol. Because of the plethora of protocols available in the IoT,[17] it would be impractical to equip every gateway with all existing adapters at deployment time. Instead, gateways can pull adapters from the controller on demand at runtime based on their context and connected clients.

Monitoring Plane

Monitoring a network's topology and QoS is a key element for edge-enabled message-oriented middleware. It is particularly important for determining proximity between clients and brokers, as proximity is calculated from network metrics such as latency, routing hops, or physical measurements such as signal strength. Both gateways and brokers need to be able to measure and report on their surrounding network state. We discuss proximity detection, and QoS and demand monitoring in more detail in Elastic Diffusion.

Control Plane

A main function of the osmotic controller is to provision brokers to edge resources, and coordinate client diffusion. To that end, the controller issues control commands via the control plane to brokers and gateways. Brokers and gateways provide control endpoints that accept coordination commands, such as an instruction to reconnect to a different broker. We discuss provisioning and network reconfiguration in more detail in Elastic Diffusion.

Elastic Diffusion

As we have established, Cloud-based message brokering cannot satisfy the stringent QoS requirements of IoT applications. This particularly concerns device-to-device communication in close proximity. Instead, brokers at the edge should facilitate low-latency communication for devices in close proximity, and forward messages to the cloud for further dissemination. This already works well for simple configurations and static infrastructure. For example, Mosquitto can be used to statically forward specific topics from one broker to another. However, as our motivational scenario illustrated, edge computing applications require a dynamic system that can adapt to changing network QoS and topology, while still allowing centralized configuration and management. During runtime, when clients at the edge require low-latency communication, the broker network needs to be expanded by provisioning brokers to edge resources dynamically. The routing topology needs to be updated, and clients in close proximity need to be migrated to the new broker. When client communication stops, these brokers can be discarded, and the network may shrink again. We call this process elastic diffusion. Figure 2 illustrates the concept.

Starting from a single configuration in the Cloud, i.e., an osmotic controller and a root broker, the system grows and shrinks the broker network during runtime to adapt to changing network topology, client mobility and demand, and resource availability. Brokers diffuse to edge resources when there is potential to optimize client QoS. Similarly, clients diffuse to edge brokers to reduce end-to-end latencies, and to balance load between brokers. A key mechanism to facilitate this is a concept we call osmotic pressure.

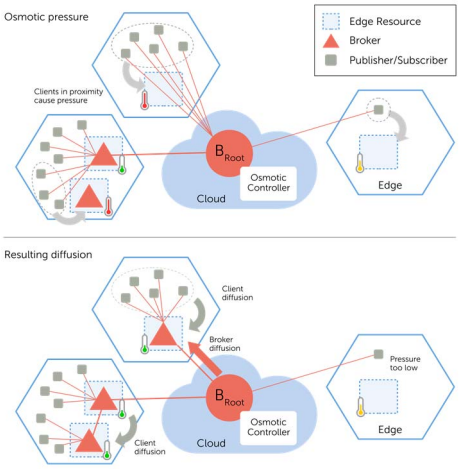

Figure 2. Osmotic pressure facilitates controlled diffusion of brokers and clients to the edge. Clients exert osmotic pressure on local resources depending on their proximity and demand. The osmotic controller aims to balance osmotic pressure throughout the system.

Osmotic Pressure

To facilitate elastic diffusion, we first need an aggregated quantification of proximity and demand. To that end, we define a metric inspired by a core concept of osmosis: osmotic pressure. Similar to the biological process, in our system, osmotic pressure causes brokers to diffuse onto

edge resources. Clients in close proximity to edge resources cause a "pulling" effect on those resources. When the pressure exceeds or drops below a given threshold, the system deploys or removes brokers from these resources. Similarly, brokers deployed on edge resource cause a "pulling" effect on clients in close proximity, which causes a reconfiguration of client-broker connections to optimize QoS for those clients. For edge resources, we calculate osmotic pressure based on the amount of clients and their proximity to the resources. Osmotic pressure is high when many clients exist in close proximity to a resource. The exerted pressure is increased with the message throughput of these clients, i.e., if local demand is high.

Proximity Detection & Demand Monitoring

To effectively calculate osmotic pressure, we need 1) a good quantification of the proximity between brokers and resources, 2) a scalable way to continuously monitor this proximity, and 3) a way to monitor demand of clients. In other distributed systems, such as Content Distribution Networks (CDN), closeness between nodes is often defined and determined by geo location, routing hops, or round-trip times (RTT). For 1), we argue that, because the main goal is to optimize end-to-end latency between clients, using the effective network latency is the best way of determining proximity between clients and resources. Additionally, for determining proximity between clients and brokers, brokers can add their average message buffering delay to the effective network latency to calculate a more precise RTT value. After all, a "close" broker on a congested resource can behave as if it were physically much farther away. However, in contrast to CDN, where proximity is determined on a per-request basis via DNS probing, osmotic MOM needs to continuously monitor the effective network latency between all nodes, which introduces serious scalability challenges. Simple monitoring solutions can cause high network overhead that becomes difficult to manage[15] For 2), we therefore propose a synthesis of network location approaches[19] with concepts such as interest management, where the frequency of monitoring is concentrated to relevant regions. For 3), a combination of throughput monitored from brokers and gateways can be used to estimate demand for a specific region, topic or content type.

Broker Diffusion

Like in osmosis, where solute molecules exert osmotic pressure, clients in close proximity of edge resources exert osmotic pressure on these resources, causing an imbalance in the network. To balance the osmotic pressure, the osmotic controller reacts by deploying a broker to the resource. We assume that edge resources provide some form of container-based virtualization, e.g., Docker, which allows the controller to deploy new brokers as containers. The new broker is integrated into the network and the controller reconfigures the routing overlay, i.e., dynamically creates topic bridges to forward messages to the cloud or other brokers if necessary. When the osmotic pressure drops, e.g., because clients move to other locations or leave the network, brokers may be discarded by the system.

Client Diffusion

Clients' connections are migrated into edge regions based on their proximity to brokers deployed at the edge. If brokers are deployed at the edge due to osmotic pressure, clients will diffuse there. Because osmotic pressure is calculated using both proximity and broker performance indicators, we have a twofold effect: clients will experience better end-to-end latencies as link usage is minimized, and load is balanced between brokers that provide similar QoS to those clients. When brokers are discarded because of low osmotic pressure, or because of a broker outage, clients diffuse back to a broker closer to the cloud.

Message Delivery Agreement

When it comes to message delivery guarantees, there are always trade-offs between the level of consistency and the end-to-end latency a system can provide, between transmission reliability and bandwidth utilization, and between transmission delay and processing requirements and se-

quencing. We recognize that even a single application can have several types of consistency demands. Take for example an analytics tool in our mHealth application. For some subscribers, it may be critical to receive every single data point in the correct order and without duplicates. For others, duplicate data may not be a problem, but the requirement is ultra-low latency delivery. The MQTT protocol addresses this in part via its "quality of service" concept, which defines the type of delivery guarantee a broker provides: at least once, at most once, and exactly once delivery.

In a single-broker system, implementing these types of guarantees is relatively straight forward, as MQTT demonstrates. However, in our distributed broker system, the type of consistency that the system must provide significantly affects how brokers and clients have to be coordinated during diffusion. For example, when a subscriber to a specific topic is migrated from one broker to another, messages received by the target broker during the reconnection process may not reach the migrating client. This may be problematic for some applications, but a coordination mechanism that provides this type of consistency guarantees can be costly in terms of resource consumption and latency penalties.

To achieve more granular control over this, we propose the concept of MDA that are negotiated between clients and brokers. An MDA is a type of Service Level Agreement (SLA) regarding message delivery. It controls how much effort the broker network makes to deliver messages between clients for a given type of topic or content, and provides the client with an estimate of the incurring latency penalty.

MDAs have two parameters: a consistency guarantee and a latency estimate. The consistency guarantee is closely related to MQTT's message delivery QoS: i.e., defines an "at least once", "at most once", or "exactly once" delivery semantic. The latency estimate is calculated from the current state of the network, the current broker workload, and the latency penalties caused by providing increased consistency. When a client requests a consistency guarantee for a specific type of topic or content, the broker responds with whether it can provide this guarantee, and if so, an estimate of how long message dissemination will take under this type of guarantee. This mechanism helps fine-tune an application's response time for clients, allows clients to handle the trade-off between latency and consistency, and allows the system to optimize coordination. When highly consistent delivery is required, the system must make extra efforts to maintain consistency during broker or client diffusion.

CONCLUSION

How to best achieve seamless Edge/Cloud integration to fully enable the Internet of Things is still subject of debate. The increasing interest in Edge computing has led to new computing paradigms that disrupt current approaches for IoT systems. One such paradigm is osmotic computing. We have shown how principles of osmotic computing can be applied to message-oriented middleware, to provide a distributed network of brokers that elastically diffuse to edge resources on demand. Unlike Cloud-based IoT platforms that typically completely neglect the proximity of devices, osmotic MOM elevates proximity to be of primary concern, both for deploying new brokers, and to optimize responsiveness for clients. Our architecture can serve as stand-alone messaging middleware to facilitate device-to-device communication in IoT environments, and can also complement complex edge computing applications that rely on message brokers, such as distributed real-time data analytics applications.

REFERENCES

1. P.T. Eugster et al., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003.
2. *AWS IoT*, Amazon; azure.microsoft.com/en-us/services/iot-hub.
3. *Azure IoT Hub*, Microsoft; azure.microsoft.com/en-us/services/iot-hub.
4. W. Shi and S. Dustdar, "The Promise of Edge Computing," *Computer*, vol. 49, no. 5, 2016, pp. 78–81.

5. M. Villari et al., "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," *IEEE Cloud Computing*, vol. 3, no. 6, 2016, pp. 76–83.
6. M. Nardelli et al., "Osmotic Flow: Osmotic Computing + IoT Workflow," *IEEE Cloud Computing*, vol. 4, no. 2, 2017, pp. 4–2.
7. A. Morshed et al., "Deep Osmosis: Holistic Distributed Deep Learning in Osmotic Computing," *IEEE Cloud Computing*, vol. 4, no. 6, 2017, pp. 22–32.
8. G. Lewis et al., "Tactical Cloudlets: Moving Cloud Computing to the Edge," *Proceedings of the 2014 IEEE Military Communications Conference*, 2014, pp. 1440–1446.
9. P.R. Pietzuch and J. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," *Proceedings of the 22nd International Conference on Distributed Computing Systems* (ICDCSW 02), 2002, pp. 611–618.
10. E. Fidler et al., "The PADRES Distributed Publish/Subscribe System," *Feature Interactions in Telecommunications and Software*, 2005, pp. 12–30.
11. M. Garcia, "How to Bridge Mosquitto MQTT Broker to AWS IoT," *The Internet of Things on AWS -- Official Blog*, blog, 2016; https://aws.amazon.com/blogs/iot/how-to-bridge-mosquitto-mqtt-broker-to-aws-iot.
12. *HiveMQ*; www.hivemq.com.
13. *JoramMQ, a distributed MQTT broker for the Internet of Things*, white paper, ScalAgent, 2014.
14. J. Gascon-Samson, J. Kienzle, and B. Kemme, "MultiPub: Latency and Cost-Aware Global-Scale Cloud Publish/Subscribe," *Proceedings of the 37th IEEE International Conference on Distributed Computing Systems* (ICDCS 17), 2017, pp. 2075–2082.
15. T. Rausch, S. Nastic, and S. Dustdar, "EMMA: Distributed QoS-Aware MQTT Middleware for Edge Computing Applications," *IEEE International Conference on Cloud Engineering*, 2018.
16. K. An et al., "An Autonomous and Dynamic Coordination and Discovery Service for Wide-Area Peer-to-peer Publish/Subscribe: Experience Paper," *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, 2017, pp. 239–248.
17. J. Weinman, "The 10 Laws of Fogonomics," *IEEE Cloud Computing*, vol. 4, no. 6, 2017, pp. 8–14.
18. A. Al-Fuqaha et al., "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, 2015, pp. 2347–2376.
19. B. Wong, A. Slivkins, and E.G. Sirer, "Meridian: A Lightweight Network Location Service Without Virtual Coordinates," *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications* (ACM SIGCOMM), vol. 35, no. 4, 2005, p. 85.

ABOUT THE AUTHORS

Thomas Rausch is a PhD student and Research Assistant at the Distributed Systems Group at TU Wien, Austria. His research interests include Internet of Things, Edge computing, and event-based systems. Rausch has a master's in Software Engineering & Internet Computing from TU Wien. Contact him at t.rausch@dsg.tuwien.ac.at or dsg.tuwien.ac.at.

Schahram Dustdar is a full professor of computer science heading the Distributed Systems Group at TU Wien, Austria. His work focuses on Internet technologies. He is an IEEE Fellow, a member of the Academy Europeana, and an ACM Distinguished Scientist. Contact him at dustdar@dsg.tuwien.ac.at or dsg.tuwien.ac.at.

Rajiv Ranjan is a reader in the School of Computing Science at Newcastle University, UK; chair professor in the School of Computer, Chinese University of Geosciences, Wuhan, China; and a visiting scientist at Data61, CSIRO, Australia. His research interests include grid computing, peer-to-peer networks, cloud computing, Internet of Things, and big data analytics. Ranjan has a PhD in computer science and software engineering from the University of Melbourne. Contact him at raj.ranjan@ncl.ac.uk or http://rajivranjan.net.