The Fog of Things Paradigm: Road toward On-Demand Internet of Things

Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, and Xiang Zhang

The authors introduce the concept of FoT, a paradigm for on-demand IoT. On-demand IoT is an IoT platform where heterogeneous connected things can be accessed and managed via a uniform platform based on real-time demands. Realizing such a platform faces challenges including heterogeneity, scalability, responsiveness, and robustness, due to the large-scale and complex nature of an IoT environment.

Abstract

In this article, we introduce the concept of FoT, a paradigm for on-demand IoT. On-demand IoT is an IoT platform where heterogeneous connected things can be accessed and managed via a uniform platform based on real-time demands. Realizing such a platform faces challenges including heterogeneity, scalability, responsiveness, and robustness, due to the large-scale and complex nature of an IoT environment. The FoT paradigm features the incorporation of fog computing power, which empowers not only the IoT applications, but more importantly the scalable and efficient management of the system itself. FoT utilizes a flat-structured virtualization plane and a hierarchical control plane, both of which extend to the network edge and can be reconfigured in real time, to achieve various design goals. In addition to describing the detailed design of the FoT paradigm, we also highlight challenges and opportunities involved in the deployment, management, and operation of such an on-demand IoT platform. We hope this article can shed some light on how to build and maintain a practical and extensible control back-end to enable large-scale IoT that empowers our connected world.

Introduction

The Internet of Things (IoT) is one of the next mega-trends in the technology world. With its ability to interconnect billions of smart things on a global scale, IoT is expected to empower innumerable new services and applications that could improve human lives, including smart cities, smart health, smart homes, Industry 4.0, and so on. IoT has a huge economic impact: the global IoT market is projected to be over US$1 trillion in the early 2020s [1].

The vision of IoT is powered by billions of connected smart things, which virtualize the real world into digital data that can be transmitted, analyzed, and further utilized. However, the massive numbers of things have led to challenges for IoT. On one hand, the current IoT, built upon existing infrastructures such as cellular networks, can hardly handle and process the tremendous amount of data generated by connected things. On the other hand, IoT manageability is also challenged by the massive and heterogeneous things, and the dynamic nature of IoT where things are plugged, unplugged, moving, or failing frequently. It is hard to manage billions of connected things with fine-grained control in real time. This situation is exacerbated by the stringent requirements of IoT applications. The majority of IoT applications are real-time in nature, meaning that they are designed to analyze continuous data streams from different locations. These applications have stringent quality of service (QoS) requirements, including but not limited to computing power, latency, throughput, loss, and robustness.

IoT devices are currently accessed and managed in an ad hoc manner: most deployed things are only visible and accessible to the owner itself, and have no public access in general. The cause of this is the lack of an on-demand architecture that provides scalable and flexible management as well as uniform and universal access to IoT services. The recently proposed Cloud of Things (CoT) paradigm aims to address this issue [2]. CoT provides centralized IoT access and management in the cloud, thereby achieving a number of benefits including on-demand service, resource pooling, virtualization, and so on. However, the cloud-based solution has its limitations. First, it does not resolve the capacity challenge: massive data still needs to be transmitted to the cloud for analysis. Second, due to its long end-to-end latency, the cloud cannot easily respond in real time to frequent network dynamics and failures within vast geographical areas. Cloud-hosted applications also receive unguaranteed QoS in terms of latency, bandwidth, and security.

Fog computing is an emerging computing paradigm [3, 4]. Different from cloud computing where all computing power is aggregated at a few globally selected locations, fog computing features the deployment of geo-distributed fog nodes across all network hierarchies, and especially in the edge network. These fog nodes provide different levels of capacity and responsiveness to meet various application needs. Fog computing inherits benefits of cloud computing including elasticity and virtualization, while bringing new benefits such as early data resolution, responsive management on the edge, and improved latency, robustness, and security. However, due to cost and energy consumption issues, fog nodes typically have limited capacities, and can only serve things and applications in nearby areas. Collaboration among fog nodes enhances capacity but also complicates system management. Fortunately, cloud computing and fog computing are not incompatible in nature; in fact, they compensate each other's limitations. This has inspired us to

Digital Object Identifier:
10.1109/MCOM.2018.1701140

The authors are with Arizona State University.

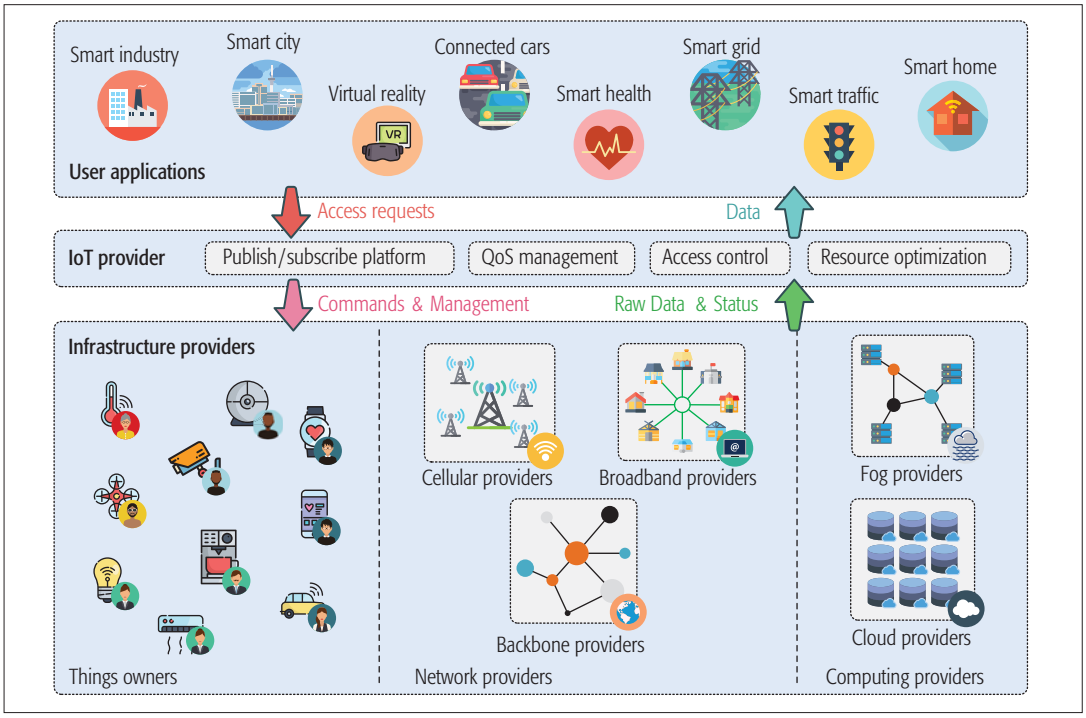

Figure 1. Three major parties in on-demand IoT: infrastructure providers, IoT provider, and users.

seek a unified approach to jointly leverage both cloud and fog computing in IoT.

In this article, we present our understanding of the direction in which the IoT shall evolve. We start from our vision on the emergence of on-demand IoT, describing its necessity and design goals. We then present an on-demand IoT paradigm that jointly utilizes fog and cloud computing, which we call the Fog of Things (FoT). FoT leverages the distributed and location-aware nature of IoT services, and features a hierarchical and reconfigurable control plane that achieves responsiveness, scalability, and other design goals. We further describe crucial challenges and opportunities in the deployment, management, and operation of FoT. We envision FoT, instead of its ad hoc or cloud-based counterparts, to be one of the cornerstones of our future smart and connected world.

Envisioning On-Demand IoT

Currently, the IoT infrastructure is mostly deployed and utilized in an ad hoc manner. Various things are produced by different vendors to fulfill similar functions. They are commonly deployed only to power a few applications that belong to the deployer/owner itself, due to lack of proper visibility and accessibility to the public. This has largely buried the true potential of IoT where myriads of distributed things generate massive data that could be used for big data analytics and decision making.

These issues have urged efforts into *on-demand IoT* (also called *service-oriented IoT* [5]), an IoT environment where functionalities (sensing, actuation, data delivery, and data analytics) can be provisioned in runtime and remotely accessed by authorized users. It is based on a concept similar to cloud computing, where computing resources are dynamically provisioned and accessed by tenants in an on-demand manner.

Three parties are involved, as shown in Fig. 1. The *IoT provider* builds and manages the IoT platform, providing various functionalities including virtualization, QoS management, resource optimization, security, and so on. *Infrastructure providers*, such as cloud providers, network providers, and things owners, participate in the business and provide infrastructure support. Finally, *users*, or *tenants*, access and utilize the provided IoT services to develop IoT applications. In general, an on-demand IoT can be built as an overlay upon existing computing and network infrastructures, but can also be built or incremented with self-owned equipment of the providers.

On-demand IoT provides benefits similar to those of cloud computing. From the business perspective, on-demand services could largely reduce capital expenditures (CAPEX) of users by reusing existing infrastructure. A well managed on-demand IoT can also reduce operational expenditures (OPEX). For things owners, allowing public access can boost resource utilization, thus increasing their utilities or revenues. For computing or network providers, on-demand IoT enables more flexible pricing options such as pay-as-you-go, which also help increase revenue. From the technical perspective, centralized management helps both infrastructure providers and users. Infrastructure providers can alleviate their overloaded components by employing smart resource allocation. Users receive guaranteed services via service level agreements (SLAs). Finally, a widely accessible IoT platform is important to the entire IoT community, inspiring collaborative technological innovations.

On-demand IoT is a diverse, large-scale, complex, and dynamic environment to build, operate, and maintain. IoT is naturally distributed, and mainly offers location-based services. Hence, the same centralized control as in cloud computing is basically not practical. Handling device heterogeneity and dynamicity requires both responsiveness

On-demand IoT provides benefits similar to those of cloud computing. From the business perspective, on-demand services could largely reduce capital expenditures (CAPEX) of users by reusing existing infrastructure. A well managed on-demand IoT can also reduce operational expenditures (OPEX).

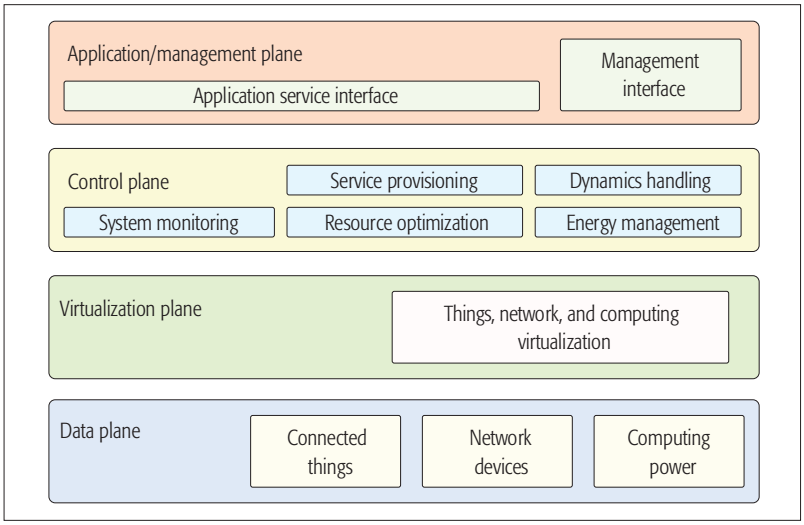

Figure 2. Four basic planes of FoT.

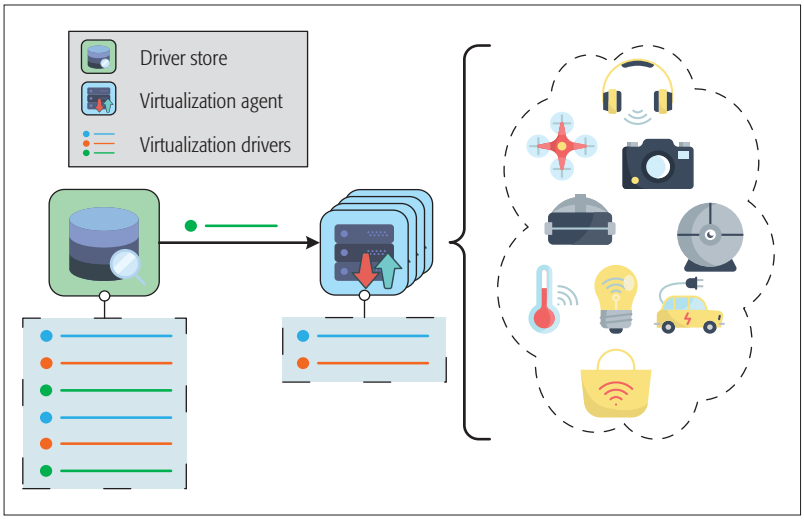

Figure 3. Virtualization plane design with three main elements: the driver store, virtualization agents, and virtualization drivers.

and scalability. In general, system management and optimization are a great challenge both on the infrastructure and on the architectural design. Below, we highlight some design goals of on-demand IoT.

Scalability is probably the most important property of any IoT system. Even a mid-scale IoT needs to manage millions of heterogeneous devices.

Virtualization is crucial in realizing dynamic and elastic services. In general, devices should be exposed only to the minimum extent to which their functions can be utilized. Devices with similar functions should be further abstracted using a uniform interface for simple and efficient access.

Responsiveness is more important in the IoT environment than in other environments, since a significant portion of IoT applications are time-critical. Responsiveness is also crucial in handling system dynamics such as device joining or removal, failure, and mobility.

Location awareness is in the nature of most IoT services. A well-designed IoT system should provide location awareness support to applications, and in return utilize it to improve system performance and management.

Robustness is to ensure system functionality during system disturbances such as failures or maintenance. Realizing robustness is specifically crucial in versatile environments like IoT, where disturbances happen frequently.

Elasticity means providing proper scaling and reconfiguration when demands from users change over time. It also means the system can sustain short-term load variations without severe congestion.

Security in IoT is different than in other environments, mainly because of the constrained nature of IoT devices. Providing native security support to resource-constrained devices is thus an important factor in architectural design. IoT security is vital, since a security breach in IoT can be much more devastating and life-threatening given IoT's ability to monitor and manipulate physical objects.

In the following, we present the design of FoT, an on-demand IoT paradigm. FoT is able to natively achieve several design goals, and also supports realizing the other goals with orthogonal technologies.

The FoT Paradigm

Architecture Overview

Our FoT architecture has four planes, as shown in Fig. 2.

The *data plane* consists of the physical components, including connected things, network devices, and computing nodes. These components perform their functionalities based on upper-plane commands. Due to heterogeneity and dynamicity, the data plane commonly requires frequent reconfiguration and optimization from upper planes.

The *virtualization plane* stands as an intermediary between physical components and decision units in the upper planes. It abstracts heterogeneous physical components into uniform and manageable virtual components.

The *control plane* is the decision core of the architecture. It performs all decision-making tasks in FoT, including component registration, service provisioning, status monitoring and reporting, failure handling, and many more. Our FoT control plane specifically features a recursively built hierarchy of controllers to achieve several design goals of FoT.

The *application/management plane* provides external interfaces of the entire FoT system, and consists of the service interface and the management interface. This plane discloses system services and parameters to authorized users/management teams, and receives application requests and system objectives to be realized by the underlying planes.

Data Plane Operation

The data plane consists of the physical components. FoT specifically features the integration of geo-distributed fog nodes, which, in addition to enhancing application performance, also enables local management of other components, as detailed later.

Two key characteristics of the data plane are *heterogeneity* and *dynamicity*. Heterogeneity causes difficulty in automatic management,

and can lead to expensive manpower for manual reconfiguration. To address heterogeneity, we add the *virtualization plane* between the conventional control plane and data plane. Dynamicity causes a scalability issue and performance fluctuation. We delegate the handling of these dynamics to the control plane to achieve fast and optimized responses.

Virtualization Plane Operation

The virtualization plane's goal is to hide the heterogeneity of the data plane. Specifically, for components with similar functions, the system maintains a general *functional template*, which defines the minimum necessary information needed to access and utilize the components. Using the functional template, the system will generate a *functional profile* for each component to describe its function, location, capacity, and other information. For example, the functional template of a surveillance camera includes its resolution, color profile, location, output format, commands, and so on. These attributes are shared by all surveillance cameras, and hence can be abstracted. The network is commonly abstracted using software-defined networking (SDN). Computing power is commonly abstracted as virtual machines (VMs).

Connected things are harder to virtualize than networks and computing, as they can be very diverse in functions and specifications. We design the virtualization plane shown in Fig. 3, which consists of three basic elements: the *virtualization drivers*, the *driver store*, and the *virtualization agents*. For all similar components (e.g., the same series of sensors of a vendor), the system maintains a *virtualization driver*, that is, a module that translates a component-specific profile into a functional profile of the component. All virtualization drivers are stored in the *driver store*, a centralized database. When the platform introduces a new type of component, for example, a new sensor model, the corresponding driver is added to the store by the IoT provider. Component virtualization is automatically performed by *virtualization agents*. Each agent keeps a list of locally stored drivers. When a new component is connected, its information is sent to the nearest agent, who will search its local list for the corresponding driver. If the driver is not available, the agent will download it from the central driver store. The agent then performs virtualization for the component as well as subsequent same-type components. These agents are distributed in the network, such as alongside controllers or at access points. They act as local bridges between the heterogeneous data plane and the uniform control plane.

Control Plane Operation

The control plane implements all the management and optimization functionalities. We propose a novel hierarchical control plane that utilizes the in-network computing power provided by fog computing to resolve the control plane scalability problem.

Hierarchical Structure: Our FoT control plane features a hierarchy of controllers that apply control over data plane components in a large geographical area, as shown in Fig. 4. The controllers are organized into a tree structure. At the bottom are leaf controllers, each covering a certain area of connected things and other components (routers, fog nodes, etc.). For example, a leaf controller can control all components in a smart building or a smart home. On top of that, several adjacent low-level controllers are aggregated and controlled by a parent controller. The parent controller also controls leftover areas between its children's controlled areas. Each controller is located at a computing node within or near the area it controls, which has sufficient computing power to support the controller's operation. The root controller aggregates global information and is commonly located in the cloud.

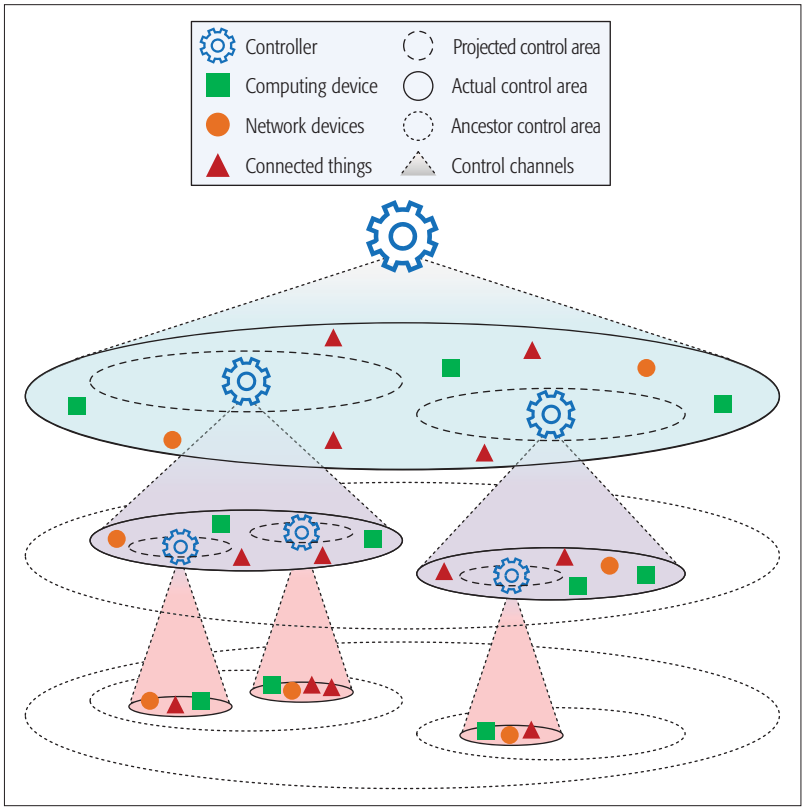

Figure 4. Our control plane design with hierarchical and recursive controllers. Dashed circles show the projected control area of a child controller of the current level, while dotted circles show the control area of ancestor controllers of current-level controllers. Child controllers are deployed in dense areas to alleviate parent load and/or provide better responsiveness.

Recursive Operation: Controllers operate in a recursive manner. Each controller (except the leaf controllers) applies both *direct* and *indirect* control to its control domain. Specifically, for components within its child controller's area, the parent controller indirectly issues queries and commands via the child. For example, if a new device access request is received at the parent controller, the request will be passed to the corresponding child controller, who will process the request and provide the corresponding access to the device if the request is authorized. For components not covered by any child, the controller directly queries and commands the components. The internal logic of our design is to ensure that *control tasks are handled by the lowest possible level of control*. For example, routing between devices within a smart building can be directly handled by the leaf controller of the building without referring to higher-level controllers.

An IoT system cannot be built in one night. Incremental deployment enables early utilization of system services, while augmenting the system with new equipment and services during normal operation. Hence the system can be scaled based on user needs.

Self-Contained Reconfiguration: Similar to the recursive architecture for cellular networks [6], the FoT control plane is reconfigurable. Controller assignment is based on the density of components within an area, and can be reconfigured by the parent controller on the fly. Furthermore, we argue that controllers should be designed to be *self-contained*. This means that a parent controller can automatically deploy and configure new child controllers at emerging dense areas in its control domain, using its controlled fog computing power, without human intervention. This enables a self-organizing control plane that can automatically adjust to system load, which is very important in achieving scalability, responsiveness, robustness, and elasticity.

Benefits: The benefits of our design are several. First, it achieves scalability by utilizing the location awareness of IoT services, reducing the states stored at higher-level controllers. For example, the root controller does not need to store the statuses of most individual devices. Each controller now works on a limited local view of the whole network, which greatly increases the overall scalability of the system. Second, our design improves responsiveness and robustness when facing network dynamics. When a component moves or fails, this event is immediately handled by the direct controller. In the case of system-wide optimization, the controller hierarchy can be configured to participate in distributed optimization, which amortizes the control overhead of using a single controller.

Application/Management Plane Operation

The application/management plane has two parts: the *service interface* and the *management interface*. The service interface exposes IoT services to tenants, including sensing and actuation by connected things, connectivity, and fog and cloud computing. It accepts application requests from tenants, and translates them into services that will be accommodated by the control plane. Similarly, the management interface exposes system status to the IoT provider. The IoT provider can specify policies through the interface that will be enforced by the control plane. This enables efficient management without frequently diving into system details.

System Function Use Cases

Component Registration: Components need to be registered to be visible and accessible. Component registration and management should be handled close to the edge to alleviate overhead at higher-level controllers. Initially, the component broadcasts a hello message. The message is sent to the nearest virtualization agent following default network rules. Upon reception, the agent looks up or downloads the virtualization driver and generates the functional profile of the component. This information is sent to the direct area controller, who creates a *virtual identity* of the component containing its uniform resource identifier (URI) and functional profile. In subsequent operations, the controller will command the component based on its functional profile. Note that the binding among component, agent, and controller can be reconfigured in runtime. For example, if the component moves, a new virtualization agent may take charge and report to a new controller.

Service Query: There are two ways an application can request a service. First, if the application knows the service URI, a direct request can be issued through the service interface. The request will be broadcast to the entire control plane. If the service is found, all controllers along the broadcast path will jointly establish routing between the servicing component and the application. Second, the application can request a generic service (e.g., video surveillance) at or near a specific location. In this case, the request will be shipped to the direct controller of the location of interest along the control hierarchy, who will then query its local components and find one that fulfills the request. In general, direct queries can be inefficient due to the need for searching the system. To reduce the overhead of direct queries, the URI can be designed to encode location information. However, this requires mobility management at the addressing level [7].

Mobility Management and Failure Handling: FoT handles network events in two steps. First, the event is immediately tackled by a local controller for fast response. For example, a leaf controller will quickly handle a local failure by finding local alternatives to avoid service disruption, such as finding replacement sensors to cover the area of a failed sensor, or alternative routing paths to bypass a router failure. Second, if the event has impact exceeding the local area, or alternatives cannot be found locally, the event will be reported to upper levels for further coordination. For example, a regional failure may involve resolution of multiple controllers.

Note that our architecture is naturally robust against control plane failures. When a child controller fails, its controlled area is automatically handed over to the parent, which ensures the continuity of system operation. This way, the only single point of failure is the root controller, which can be backed up by replicas in the cloud. Still, events are handled at the lowest possible level of control, which ensures responsiveness and scalability of the system.

Challenges and Opportunities

Deployment

An IoT system cannot be built in one night. Incremental deployment enables early utilization of system services, while augmenting the system with new equipment and services during normal operation. Hence, the system can be scaled based on user needs. There are several factors that need to be considered for system incrementation. First, deploying new devices incurs various costs including deployment, management, energy, and so on. Second, investment into different dimensions (new connected things, network devices, or computing nodes) at different locations may result in different improvements of system performance. Therefore, it is advised that the IoT provider optimize its deployment utility based on system-wide measurement of performance.

Management

Runtime Control Plane Reconfiguration: One key feature of our design is that controllers can be automatically deployed or revoked based

on data plane load. This ensures elastic control against fluctuating loads. However, deploying controllers can be costly, especially in dense areas where computing power is already scarce. In this case, the system needs to consider the trade-off between deploying more local controllers to improve system manageability or devoting more computing power and energy to improving application performance. Finding the optimal deployment and assignment policy subject to capacities and dynamic loads constitutes an optimization problem to be addressed.

Network Planning and Orchestration: In IoT, the network is largely the performance bottleneck due to its limited capacity and long delay. Network planning techniques such as QoS-aware routing [8], traffic engineering [9], and interference management [10] have each demonstrated their advantages in different network environments. However, applying these techniques in IoT incurs scalability and dynamicity issues. To resolve the scalability issue, proper traffic classification and/or aggregation is needed to reduce control plane states. To resolve dynamicity, both offline and online algorithms need to be developed; the former achieves resource planning in the long run, while the latter provides quick responses to network dynamics. Furthermore, service function chaining should be considered during network planning, which constitutes the network orchestration problem.

Service Provisioning and Orchestration: Service provisioning fulfills application service requests. Services are provisioned in several dimensions: connected thing access, data delivery, and data processing. These can be considered either separately or jointly. For example, a real-time processing application that analyzes data streams from distributed sensors would require joint consideration of sensor data access, data delivery, (potential) in-network pre-processing, and analytics logic embedding. Factors to be considered include usage costs, network and computing power, energy consumption, QoS (bandwidth, latency, etc.), robustness, elasticity, multi-tenant resource sharing, and so on. An important consideration is to utilize geo-distributed fog nodes to host data processing and analytics, to achieve early resolution of the massive data at the edge. A related problem is service orchestration [4], where an application is decomposed into distributed sub-services. Optimization algorithms can be developed, but a general framework that can incorporate different dimensions and constraints is preferred.

Energy Management and Optimization: Energy management is a crucial part of IoT [11]. First, a large number of connected things are battery-powered, and hence have very tight energy budget. Second, deployment of connected things tend to be denser in areas that are already congested with devices (e.g., business districts, urban centers, and factories). Energy consumption of massive things could cause problems for the energy grid that serves other more critical services, such as lighting or emergency systems. The use of energy harvesters could alleviate the situation, but are not available in scenarios like indoor environments. Proper energy management should jointly consider energy consumption of all components, and utilize various energy sources including power networks, local power generators, energy harvesters, and the smart grid. Both short-term and long-term energy planning is helpful in FoT.

Scalability: With all the flexibility of centralized management comes the concern of scalability. For example, using SDN as the network controller may suffer from the intrinsic scalability issue of SDN controllers. Our hierarchical control plane serves as a natural remedy for this issue. The IoT provider can deploy multiple levels of SDN controllers, where each controller controls a local domain, much like the way our FoT controllers work. In fact, our hierarchical control plane can be viewed as a generalization of the existing hierarchical SDN architecture [6], which has already demonstrated the scalability gain of such a design. Nevertheless, scalability will remain a problem in IoT even with such an approach, and surely deserves future research and development efforts.

With all the flexibility of centralized management comes the concern of scalability. For example, using SDN as the network controller may suffer from the intrinsic scalability issue of SDN controllers. Our hierarchical control plane serves as a natural remedy for this issue.

Security

Constrained Device Security: One major challenge in IoT security is the constrained nature of IoT components. Components such as battery-powered sensors or RFID tags have very limited computing power and energy budget [12], and thus are hardly capable of running complex cryptographic algorithms. With the emergence of IoT-related attacks [13], development of constrained security mechanisms is both important and urgent. However, this field of study is still in its infancy and requires extensive efforts in the near future.

Infrastructure-Assisted Security: One way of realizing effective but inexpensive IoT security is to rely on the platform itself. Such practices have already been utilized in other environments like data centers and backbones: by deploying security functions within the network, traffic can be checked before reaching the end hosts. Such an approach can be extended to the IoT scenario. For example, functions that help establish secure channels at access points could greatly alleviate the resource burden on constrained devices, while still receiving most of the benefits of secure transmission. Despite some early efforts [14], there has not yet been much research in this field. We anticipate that infrastructure-assisted security will play a significant role in IoT security.

Privacy: Privacy is of significant concern in IoT, since a majority of IoT applications are based on location [15]. While existing location privacy mechanisms address this issue for each single service or application, using a combination of different location-based services could still leak sensitive information. This again requires privacy mechanisms natively embedded into the platform instead of handled by things owners or application developers. IoT can also invade private spaces like homes or factories, which could cause leakage of sensitive information other than location. Proper protection of such information is another direction for future research.

Conclusions

In this article, we present a novel on-demand IoT paradigm, the Fog of Things. The FoT paradigm extends both the data plane and the control plane to the network edge, thus achieving many benefits including scalability, responsiveness, robust-

We envision the FoT paradigm to be a major enabler of on-demand IoT, and will play a crucial role in our smart and connected future. Enabling such a future, however, requires extensive future work on the development and implementation of the FoT framework as well as resolving its various issues such as scalability and security.

ness, location awareness, and so on. We lay out the design of the FoT architecture, which features a flat-structured virtualization plane and a hierarchical and recursive control plane. The virtualization plane achieves universal abstraction of data plane components, while the control plane achieves scalable and fine-grained control utilizing location awareness of IoT services. We explain the detailed operation of each plane and the system. We also highlight challenges and opportunities in deployment, management, and security of FoT. In general, we envision the FoT paradigm to be a major enabler of on-demand IoT that will play a crucial role in our smart and connected future. Enabling such a future, however, requires extensive future work on the development and implementation of the FoT framework as well as resolving its various issues such as scalability and security.

Acknowledgment

This research was supported in part by NSF grants 1461886, 1704092, and 1717197.

References

[1] "IoT Market Forecasts"; https: //www.postscapes.com/internet-of-things-market-size/, accessed May 15, 2018.
[2] A. Botta *et al.*, "Integration of Cloud Computing and Internet of Things: A Survey," *Future Generation Comp. Sys.*, vol. 56, Mar. 2016, pp. 684–700.
[3] M. Chiang *et al.*, "Clarifying Fog Computing and Networking: 10 Questions and Answers," *IEEE Commun. Mag.*, vol. 55, no. 4, Apr. 2017, pp. 18–20.
[4] P. Hu *et al.*, "Survey on Fog Computing: Architecture, Key Technologies, Applications and Open Issues," *J. Net. Comp. Appl.*, vol. 98, Sept.–Nov. 2017, pp. 27–42.
[5] V. Issarny *et al.*, "Revisiting Service-Oriented Architecture for the IoT: A Middleware Perspective," *Proc. ICSOC*, 2016, pp. 3–17.
[6] M. Moradi *et al.*, "SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture," *Proc. ACM CoNEXT*, 2014, pp. 377–90.
[7] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy, "DART: Dynamic Address RouTing for Scalable Ad Hoc and Mesh Networks," *IEEE/ACM Trans. Net.*, vol. 15, no. 1, Feb. 2007, pp. 119–32.
[8] J. W. Guck *et al.*, "Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation," *IEEE Commun. Surveys & Tutorials*, vol. 20, no. 1, 2018, pp. 388–415.
[9] A. Mendiola *et al.*, "A Survey on the Contributions of Software-Defined Networking to Traffic Engineering," *IEEE Commun. Surveys & Tutorials*, vol. 19, no. 2, 2017, pp. 918–53.
[10] M. Noura and R. Nordin, "A Survey on Interference Management for Device-to-Device (D2D) Communication and Its Challenges in 5G Networks," *J. Net. Comp. Appl.*, vol. 71, Aug. 2016, pp. 130–50.
[11] N. Kaur and S. K. Sood, "An Energy-Efficient Architecture for the Internet of Things (IoT)," *IEEE Sys. J.*, vol. 11, no. 2, June 2017, pp. 796–805.
[12] H. Hellaoui, M. Koudil, and A. Bouabdallah, "Energy-Efficient Mechanisms in Security of the Internet Of Things: A Survey," *Comp. Networks*, vol. 127, Nov. 2017, pp. 173–89.
[13] S. Cobb, "10 Things to Know About the October 21 IoT DDoS Attacks"; https://www.welivesecurity.com/2016/10/24/10-things-know-october-21-iot-ddos-attacks/, accessed May 15, 2018.
[14] S. Nair, S. Abraham, and O. A. Ibrahim, "Security Fusion: A New Security Architecture for Resource-Constrained Environments," *Proc. USENIX HotSec*, 2011, p. 2.
[15] L. Chen *et al.*, "Robustness, Security and Privacy in Location-Based Services for Future IoT: A Survey," *IEEE Access*, vol. 5, 2017, pp. 8956–77.

Biographies

Ruozhou Yu [S'13] received his B.S. degree from Beijing University of Posts and Telecommunications, China, in 2013. He is now a Ph.D. candidate in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. His research interests include network virtualization, software-defined networking, cloud and data center networks, edge computing, the Internet of Things, and so on.

Guoliang Xue [M'96, SM'99, F'11] is a professor of computer science and engineering at Arizona State University. He received his Ph.D degree (1991) in computer science from the University of Minnesota, Minneapolis. His research interests include survivability, security, and resource allocation issues in networks. He is an Editor of *IEEE Network* and the Area Editor of *IEEE Transactions on Wireless Communications*, Wireless Networking.

Vishnu Teja Kilari [S'13] received his M.S. degree from Arizona State University, Tempe, in 2013. Currently he is a Ph.D. student in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. His research interests include botnets, smart grid security and hardware assisted security.

Xiang Zhang [S'13] received his B.S. degree from the University of Science and Technology of China, Hefei, in 2012. He is now a Ph.D. candidate in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. His research interests include network economics and game theory in crowdsourcing and cognitive radio networks.