Practical Fog Computing With Seattle

Albert Rafetseder
NYU Tandon School of Engineering
albert.rafetseder@univie.ac.at

Lukas Pühringer
NYU Tandon School of Engineering
lukas.puehringer@nyu.edu

Justin Cappos
NYU Tandon School of Engineering
jcappos@nyu.edu

***Abstract*—In this paper we present Seattle, a practical and publicly accessible fog computing platform with a deployment history going back to 2009. Seattle's cross-platform portable sandbox implementation tackles the widely-recognized issue of node heterogeneity. Its componentized architecture supports a number of approaches to operating a Seattle-based fog system, from isolated, standalone and peer-to-peer operations, to full-fledged provisioning by a dedicated operator, or federations of many operators. Seattle's components and interfaces are designed for compatibility and reuse, and may be aligned with existing trust boundaries between different stakeholders.**

Seattle comprises implementations of all components discussed in this paper. Its free, open-source software stack has been used for teaching and research; outside groups have used existing Seattle components, and constructed new components with compatible interfaces in order to adapt the platform to their needs.

I. Introduction And Related Work

In recent years, fog computing has established itself as an approach to cloud computing at the edge of the network. Fog systems are typically characterized by the large number of geographically distributed nodes, ranging from embedded systems, like network connected sensors, to smartphones and end-user laptops [6], [19], [41]. While leveraging the ubiquitous availability and specific capabilities of such devices brings many opportunities, it also carries new challenges unknown to traditional cloud services. Such challenges include the heterogeneity of fog nodes [6], [28], [34], [41], their accessibility, e.g. behind private networks, and extended security and privacy requirements [7].

Another issue is that existing proposed fog computing architectures have been described as "siloed" [4], referring to the lack of open interfaces that characterize today's fog devices and infrastructures. Furthermore, opportunities and challenges have been widely addressed in the respective literature from a theoretical point of view, but actual fog computing implementations — let alone actively deployed and publicly available platforms that can be used in fog contexts — are rare.

In this paper, we present our Seattle platform and make the case for why its unique design is well suited for fog computing applications. But, to provide context, we first review relevant aspects of the existing literature.

A. Related Work

Emerging technologies have opened potentially promising new areas for research in fog computing. Bellavista et al. [3] investigate the applicability of Docker containers for fog computing. The work uses and extends the open-source Kura framework to create Internet of Things (IoT) gateways that control information flow between fog nodes and the cloud, while also reviewing and benchmarking different container-related technologies used in these nodes. The focus of this work is on IoT computing, where tailored services run on fog nodes to gather data that is forwarded to the cloud for further processing. A similar evaluation of Docker containers for edge computing can be found in [26].

Others have contributed fog platforms for very specific use cases, e.g. an emergency alert system using smartphones that propagate alerts to nearby emergency departments [1], an idea also proposed by Masip-Bruin et al. [29]. Gazis et al. introduce an "Adaptive Operations Platform" to effectively apply equipment failure models within the context of Industrial IoT [23]. Amrutur et al. [2] discuss the use of a testbed for "smart city" apps that could be mounted on light poles. Vehicular use cases, including ones that are based on ad-hoc networks, are presented by Bitam et al. [5] and Truong et al. [37].

Apart from individual use cases, some implementations address specific individual fog computing issues, like Dsouza et al. who propose a policy management framework to authenticate the various actors in applications like smart transportation systems [20]. Yi et al. [40] point out vendor lock-in as a possible problem for fog computing, and attempt to exploit locality in a real-world experiment with face recognition software.

In addition to academic work, commercial business solutions using fog computing have also evolved. Cisco IOx [17] promotes a system that allows traditional Linux application development on Cisco IOS powered networking devices. Google recently released "Google IoT cloud," a system for cloud based device management and a protocol bridge to connect edge nodes to cloud analytic systems [24] and other infrastructure.

Finally, there are different stances on operational aspects of fog computing platforms. Some authors assume a centrally-managed and operated approach, perhaps including monetarization [28], [34], whereas others [4] call for open interfaces and interoperability so that many fog environments can coexist and provide federated services.

As will be discussed shortly, our platform Seattle offers a practical, useable sandbox implementation that tackles the widely-acknowledged node heterogeneity issue in fog computing. Furthermore, Seattle's components are designed for loose coupling and precise trust boundaries, so as to enable multi-stakeholder operations. This enables out-of-the-box deployments with minimal mutual trust requirements. All the

Seattle components described in this paper are active, live, and publicly accessible. All software is Free, Open-Source Software, and available from public repositories[1].

B. Contributions

The contributions of this paper are as follows:

- We present Seattle, a platform uniquely adaptable for fog computing research and applications. Seattle is capable of real-world deployment on heterogeneous nodes, including desktop and laptop machines, Android devices, Raspberry Pis, and routers and embedded devices running OpenWrt.
- We explain how Seattle's system architecture can cater to a variety of use cases, ranging from peer-to-peer deployments to full-fledged provisioning by a dedicated operator, to cooperative setups, where different stakeholders federate and unite multiple parallel running instances of services.
- We validate Seattle's ability to facilitate fog computing systems by presenting our live deployment. Already installed on tens of thousands of devices, it has been used by over 4,000 researchers and students during its 8 years of operational history.

Our past deployments of Seattle have served multiple contexts including research [12], [18], [21], [27], [32], [33], [38], [42], [44] and education [10], [11], [13], [25], [39]. In addition, other groups have successfully reused and augmented Seattle components for their own specific purposes [14]–[16], [22], [30], [35], [36], [43], [45].

II. System Architecture

An important insight behind the design of Seattle as a platform is how the classical dichotomy of service platform operators and users gives way to multi-faceted trust relationships between a large number of mutually-unrelated stakeholders. In Seattle, the actual edge-based software installation, core infrastructure services, clearinghouse operations, platform software builds, and remote application deployments might be carried out and managed by different groups of people. Such people are potentially untrusted (or even unknown), to members of the same group.

This gives rise to a set of components [8] — any of which is useful by itself — that can be freely combined to implement various deployment models, from pure peer-to-peer operation up to a provisioned deployment by a dedicated operator. Furthermore, new components may be introduced freely to replace or augment existing ones, as long as all adhere to the component interfaces. We briefly introduce the default component implementations below. As detailed in Section IV, we have actively used and developed these components over the last eight years as a part of Seattle's public, live deployment.

[1] https://github.com/SeattleTestbed

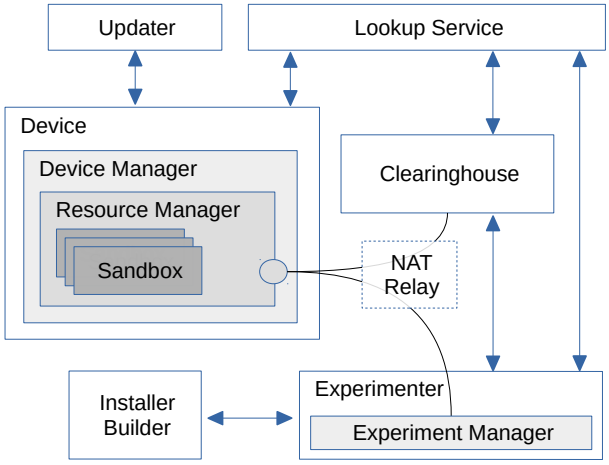

Fig. 1. Components and interconnections of Seattle's architecture.

A. Components

The core components of Seattle and their interconnections are shown in Figure 1. We first introduce the components running on all devices that run Seattle. For computation, the primary component is a restricted, performance- and security-isolated Python-based *sandbox* [9], [27]. The sandbox Application Programming Interface (API) provides networking, file, threading, and other functions to the code it executes. Note that due to Seattle's research and educational background, we call code that is executed in sandboxes an "experiment," regardless of its actual nature. In the context of fog computing, the sandbox performs distributed computing and runs code to trigger actuators and read sensors.

The *resource manager* combines the functionality of a hypervisor with a remote-control server. It isolates sandboxes that run in parallel, and interfaces with them on behalf of authorized remote parties to start, stop, or reset them, upload and download data, or transfer ownership. The sandboxes on one device can all have different authorized users if so desired, so that one Seattle node can serve multiple experimenters. Due to the heterogeneity encountered in today's networks, the resource manager also includes techniques to traverse Network Address Translation (NAT) gateways and firewalls as often found in end-user gateway routers.

The *device manager* is the device owner's interface to enable or disable a Seattle install, and choose the amount of resources that it may consume on the respective device.

The *software updater* keeps all components of the device software up to date.

An *experiment manager* controls the fundamental functions of sandboxes, such as starting the execution of code, or downloading collected data. The experiment manager programmatically contacts sandboxes through the resource manager interface, and provides a human-oriented interface to the experimenter. The experimenter authenticates against sandboxes by using cryptographic keys.

All experimenters run an experiment manager, and possess their own set of cryptographic keys. This enables interesting deployment options (and thus applications). Two experimenters could choose to mutually authenticate each other on the sandboxes they control, or swap resources, and so on. Thus, it makes more sandboxes available to an experiment than a single experimenter could ever reach. Section II-C explains in greater detail how Seattle manages trust relationships.

Seattle's device software and an initial set of cryptographic keys controlling the sandboxes are shipped in the form of a bundled installer for the specific platform Operating System (OS). The *installer builder* component is responsible for packaging and hosting the code and public keys. The installer builder allows experimenters to create their own custom installers, where they get to choose what public key or keys are to be included. Every custom installer can be referenced through its specific, static Uniform Resource Locator (URL). Thus, by selecting and installing a specific packaged installer, device owners can choose which experimenters to initially grant access to sandboxes on their devices.

The default Seattle architecture also includes a *lookup service*. This is a generic network-accessible key/value store that the resource manager and experiment manager can use to announce and retrieve information about sandboxes. For instance, a resource manager announces its contact information, such as its own or a NAT forwarder's IP address and port number, under the public keys of all currently authorized experimenters. The experiment manager retrieves this contact information by looking up the experimenter's public key at the lookup service, and then proceeds to contact every resource manager that hosts sandboxes the experimenter can use.

We use *NAT Relays* on some Seattle nodes that are accessible from the public Internet. In this manner, firewalled or NATted nodes have a common, public meeting point to which they can direct experiment managers.

Lastly, Seattle provides a *clearinghouse*, which is a centralized sandbox-sharing site. As mentioned earlier, sandboxes can be swapped. A clearinghouse can act as a trusted intermediary on behalf of all registered experimenters, thus creating a large pool of available sandboxes without requiring all experimenters to set up individual mutual trust relationships. The clearinghouse uses the same resource manager interfaces as the experiment manager, and provides a human-oriented interface through which experimenters can request and release sandboxes. The clearinghouse implements internal policies that govern the exact rules for swapping sandboxes between registered experimenters.

Our live deployment of Seattle includes implementations of all components. This also includes a clearinghouse that offers free access to an initial set of sandboxes for all registered experimenters.

B. Roles And Operational Flexibility

The preceding section implicitly introduced different roles relevant in the Seattle architecture: *device owners* control whether and how they participate in Seattle, so as to provide sandboxes to the platform; *experimenters* use these sandboxes to run their code on remote devices; and, lastly, *operators* manage various components of the infrastructure. These roles match what most existing fog computing architectures anticipate. However, the limited and clearly-defined boundaries of Seattle components result in few requirements for mutual trust or, indeed, interaction between the actual persons acting in these roles. This enables operational scenarios including, but also exceeding, the often-assumed centrally managed and operated fog computing setups.

For instance, in a local experimental deployment, the owner of all devices might also serve as an experimenter at the same time. If the experiment managers and installer builders can be reused from existing deployments, no additional effort is involved in setting up the infrastructure. A setup on exclusively-controlled, all-local devices can serve as a testbed for research, or enable device owners to run an "OwnCloud"-like fog computing setup on their equipment.

The same approach also scales to groups of device owners and experimenters who federate to share resources on (parts of) their setups. No trusted intermediary in the form of a clearinghouse is required. Trust relationships among the researchers and device owners can be set up out-of-band as well.

Another implementation could utilize all of the components described in Section II-A above. Such a deployment would start with a sponsored set of sandbox resources, a clearinghouse to mediate access between sponsored and contributed sandboxes and registered researchers, and all of the auxiliary services discussed. One such example is the current deployment of Seattle. We operate instances of all components, while further instances (particularly of sandboxes on remote nodes) are contributed and operated by volunteers. Section IV details our current deployment.

Sensibility Testbed [42] and Social Compute Cloud [14] provide a similar breadth in capabilities as Seattle. However, they only operate a partial infrastructure, and reuse much of Seattle's. Their particular clearinghouse implementations differ from Seattle's in terms of user interfaces and policy implementations. Sensibility also provides a new sandbox type that allows privacy-preserving access to sensors on Android smartphones and tablets. Other components, including the experiment manager, are reused without changes.

Finally, we envision that multiple central operators could set up their own instances of all components, and then federate using Seattle's open interfaces to barter or trade resources. Operator-specific clearinghouses serve as centralized trust anchors, and "experiments" would instead be "apps" for end users, with the former experimenters as over-the-top service or software providers.

C. Trust Management

As detailed in the previous subsection, interpersonal or institutional trust plays an important role in a Seattle-based system. For instance, the device owner will only run an installer package obtained from an installer builder that is operated

by a party they trust. Similarly, multiple clearinghouses will only cooperate if mutual trust relationships can be established.

Additionally, Seattle uses asymmetrical cryptography as a technical means to map out trust. Its core computational component, the sandbox, incorporates the roles of an *admin* and *users*, all identified by public keys added to the installer package by the installer builder. The admin of a sandbox can add or remove users, split the resources of a sandbox, or give up privileges in favor of another admin (i.e., another public key). An authorized user runs programs in the sandbox, transfers data, and so on. If more than one user is authorized for the same sandbox, they all share the same sandbox environment.

Depending on the deployment scenario realized, interpersonal or institutional and technical trust relationships align at different places through interactions with multiple components in the system. The first point is the installer builder, as the person configuring the installer package chooses the initial set of admin and user public keys going into it. Then, the two types of trust relationships align again in the experiment manager, for example when an admin adds new user public keys to sandboxes he controls. Finally, if the deployment uses a clearinghouse to administer sandbox access, this becomes the third point of alignment. The clearinghouse must trust an experimenter (i.e. sandbox user) before placing the public key in sandboxes it controls.

As a final remark, the software updater component of the Seattle installer also contains a public key. It is used to verify the integrity of updates. This key is likely managed by the operator of the installer builder who controls the software that is packaged. However, there is no intrinsic requirement for this to be the case. A trust relationship may extend to this component as well.

III. Implementation

In this section we will look at actual implementations of selected components, with a focus on those aspects that help Seattle solve the node and operational heterogeneity challenges encountered in fog computing.

A. Sandbox Implementation

Seattle's Python-based default sandbox [9] offers a cross-platform portable, resource-isolated, safe execution environment for untrusted experimenter code.

The choice of a high-level programming language environment trades off some reduction in performance against large gains in portability across OSs and devices. Seattle's platform abstractions not only provide a unified API to sandboxes on desktops and laptops, but also smartphones and even WiFi routers. Seattle sandboxes run on Windows, Mac OS X and later, Linux, and the Berkeley Software Distributions (BSDs), as well as Android and OpenWrt.

Seattle's resource isolation scheme [27] ensures that each sandbox is confined to strict usage quotas for all resources of the hosting system, including its Central Processing Unit (CPU) time and memory, used disk space, and even Internet Protocol (IP) addresses and port numbers on network interfaces.

In terms of code safety, Seattle keeps buggy or deliberately destructive experimenter code from harming the host machine. This is guaranteed by first checking the static code safety and forbidding potentially problematic statements (like importing arbitrary modules). At runtime, all of the API functions strictly check their call parameters, so that, for example, file names can be sanitized and restricted to the sandbox directory.

Other sandbox implementations have augmented Seattle's basic functionality. Sensibility Testbed adds sensor functions to Seattle's Python-based sandbox. Another internal research prototype of ours interfaces the Seattle resource manager with Docker, so that its sandboxes are Linux containers.

B. Component Interface Implementation

A core solution to flexibly addressing operational heterogeneity (see Section II-B) lies in clearly-defined interfaces that use simple, text-based, mostly stateless protocols. Such an approach has a few benefits. Code handling communications between components can be much simpler, as parsing is easy to verify, even if done manually. As such, it is much easier to reimplement these protocols to interface with Seattle from other systems. Lastly, keeping and tracking state in a distributed fashion is a notoriously difficult problem that we attempt to avoid wherever possible.

For communication that requires a token of privilege to trigger an action on the remote side, control messages contain signatures constructed with the private key of the experimenter. A sandbox controlled by that experimenter knows the corresponding public key (as it previously downloaded a Seattle installer customized to contain this key). Only after checking the message signature are privileged operations, such as starting or stopping the sandbox or transferring data to/from its local storage, permitted.

Scripted (i.e. non-interactive) communications with the clearinghouse, installer builder, and software updater all use HyperText Transfer Protocol (HTTP) or HTTP Secure (HTTPS) in the form of an XML-RPC interface, which further simplifies integration. Also, these interfaces follow what is known as Postel's Robustness Principle [31, §2.10], and are quite liberal in the data they accept from the remote side of the communication. This has practical relevance for some of the calls, as call parameters are treated as opaque and are not interpreted by the receiving instance. Thus, additional information not previously foreseen and designed into the protocol can still pass.

IV. Practical Deployment

As mentioned previously, we have running implementations of all Seattle components on nodes on the public Internet, including many instances we neither control nor operate. These latter implementations have been contributed to our clearinghouse's resource pool by volunteers. Apart from the numbers we report below, we do not have any insight on offspring projects that reuse our open-source software base,

but set up their own independent deployments. Seattle has no tracking code besides what an operating clearinghouse needs to support the sandboxes it controls.

To assess the current scale and size of Seattle, we report statistics from the IP addresses contacting our software updater service. This service is contacted by all installations that Seattle's main installer builder shipped to end-user devices, and has reported downloads from over 40,000 different IP addresses over our eight years of deployment history. This metric is not perfect: it undercounts multiple devices behind a NAT gateway, and overcounts devices that change their IP address frequently (such as mobile nodes). However, it gives an impression of the scale of our deployment.

Attempting to reverse-resolve these IP addresses to Domain Name Service (DNS) names yields no result for about 50% of addresses; resolves to names that belong to home Internet providers for another 30%; and to university machines for about 15%. The latter likely includes students, since they often use on-campus computers in classrooms or labs. Note also that the number of online nodes (and thus sandboxes) varies over time, as device owners turn off or suspend their devices as they move, or as their daily routines start and end. Figure 2 plots a world map of Seattle nodes, with the colors encoding the approximate node density in an area. Node locations were generated from GeoIP data. It shows that Seattle has notable user bases outside of the United States (where it was originally developed), particularly in Central Europe and China.

Since its inception, more than 4,000 experimenters have used Seattle's clearinghouse to request access to resources on remote devices. We attribute this to the fact that our clearinghouse offers ten free sandboxes for every registered user, and contributing one's own resources counts as a "credit" to use more remote sandboxes in turn. Resource credits have proven a valuable incentivization strategy to push the adoption of Seattle.

While offering generous resources to bootstrap experiments, Seattle has also benefited greatly from generous contributions of code, documentation, tutorials, and libraries from more than 100 contributors at 32 institutions all over the world. Seattle strives to make contributing easy. All of its components are open-source software, and thus easy to inspect, adapt, and reuse in other contexts.

Sensibility Testbed [42] has been particularly successful in adapting and reusing Seattle components. It extends Seattle's sandbox type so as to allow access to sensors on Android smartphones and tablets. Furthermore, its device manager includes an additional Graphical User Interface (GUI) in the form of a native Android app. Sensibility Testbed's clearinghouse implements special policies not found in Seattle. An experimenter requesting sandboxes on Sensibility Testbed must first complete an Institutional Review Board (IRB) approval process, which ensures that the experiment conducted does not impact the privacy of device owners involved.

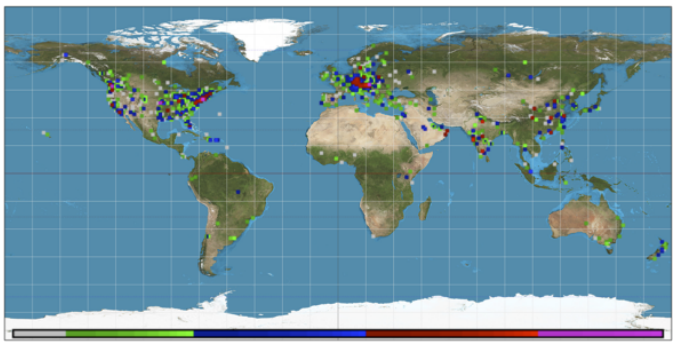

Fig. 2. World map of Seattle node locations. Color shows approximate density from low (grey) to high (magenta).

V. Conclusion And Outlook

This paper presents Seattle, a practical and publicly accessible platform for fog computing. Seattle is deployed and operational in the real world on heterogeneous nodes, including desktop and laptop machines, Android devices, Raspberry Pis, and routers and embedded devices running OpenWrt. Seattle's installer packages and sandbox implementation thus address a widely-recognized and central issue for the success of fog computing, which is node heterogeneity.

The system architecture of Seattle consists of loosely-coupled components. This makes the components useful as stand-alone entities, and stimulates reuse and adaptation of existing component implementations for unforeseen purposes. Furthermore, since components are separated along trust boundaries, Seattle does not require a single centralized operator for all components. Instead, a large range of operational scenarios with varying scopes can be implemented, from fully local setups to peer-to-peer resource swapping and federated multi-operator deployments with mutually-trusted intermediaries. Seattle's support of heterogeneous operations includes and exceeds the capabilities of often-assumed centrally managed and operated fog computing setups.

Currently, we can report an active, live, and publicly accessible deployment of all the Seattle components described in this paper. More than 4,000 experimenters have used our existing deployment to run distributed experiments, and more than 100 developers from 32 institutions have contributed to Seattle's free, open-source software stack since its inception in 2009. Seattle has been installed on 40,000 devices all over the world, and has been used for teaching and research. Furthermore, outside groups have used existing Seattle components to construct new components with compatible interfaces, and to adapt the platform to their needs. All of Seattle's software is Free, Open-Source Software and available at `https://github.com/SeattleTestbed`.

References

[1] M. Aazam and E. N. Huh. E-hamc: Leveraging fog computing for emergency alert service. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 518–523, March 2015.

[2] Bharadwaj Amrutur, Vasanth Rajaraman, Srikrishna Acharya, Rakshit Ramesh, Ashish Joglekar, Abhay Sharma, Yogesh Simmhan, Abhijit Lele, Ashwin Mahesh, and Sathya Sankaran. An Open Smart City IoT Test Bed: Street Light Poles As Smart City Spines: Poster Abstract. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pages 323–324, New York, NY, USA, 2017. ACM.

[3] Paolo Bellavista and Alessandro Zanni. Feasibility of Fog Computing Deployment Based on Docker Containerization over RaspberryPi. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN '17, pages 16:1–16:10, New York, NY, USA, 2017. ACM.

[4] L. Belli, S. Cirani, L. Davoli, A. Gorrieri, M. Mancin, M. Picone, and G. Ferrari. Design and Deployment of an IoT Application-Oriented Testbed. *Computer*, 48(9):32–40, September 2015.

[5] S. Bitam, A. Mellouk, and S. Zeadally. VANET-cloud: a generic cloud computing model for vehicular Ad Hoc networks. *IEEE Wireless Communications*, 22(1):96–102, February 2015.

[6] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[7] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, 56:684–700, March 2016.

[8] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: A platform for educational cloud computing. In *The 40th Technical Symposium of the ACM Special Interest Group for Computer Science Education (SIGCSE '09)*, 2009.

[9] J. Cappos, A. Dadgar, J. Rasley, J. Samuel, I. Beschastnikh, C. Barsan, A. Krishnamurthy, and T. Anderson. Retaining sandbox containment despite bugs in privileged memory-safe code. In *The 17th ACM Conference on Computer and Communications Security (CCS '10)*, 2010.

[10] Justin Cappos and Ivan Beschastnikh. Teaching networking and distributed systems with seattle. *J. Comput. Sci. Coll.*, 25(1):173–174, October 2009.

[11] Justin Cappos and Ivan Beschastnikh. Teaching networking and distributed systems with seattle: tutorial presentation. *J. Comput. Sci. Coll.*, 25(5):308–310, May 2010.

[12] Justin Cappos, Lai Wang, Richard Weiss, Yi Yang, and Yanyan Zhuang. Blursense: Dynamic fine-grained access control for smartphone privacy. In *Sensors Applications Symposium (SAS)*. IEEE, 2014.

[13] Justin Cappos and Richard Weiss. Teaching the security mindset with reference monitors. *SIGCSE Bull.*, 45(1), 2014.

[14] Simon Caton, Christian Haas, Kyle Chard, Kris Bubendorfer, and Omer Rana. A social compute cloud: Allocating and sharing infrastructure resources via social networks. In *IEEE Transactions on Services Computing*. IEEE, 2014.

[15] K. Chard, K. Bubendorfer, S. Caton, and O.F. Rana. Social cloud computing: A vision for socially motivated resource sharing. *Services Computing, IEEE Transactions on*, 5(4):551–563, Fourth 2012.

[16] Kyle Chard, Simon Caton, Omer Rana, and Kris Bubendorfer. Social cloud: Cloud computing in social networks. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 99–106. IEEE, 2010.

[17] Cisco iox. https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html, last access: Jun 2017.

[18] L. Collares, C. Matthews, J. Cappos, Y. Coady, and R. McGeer. Et (smart) phone home! In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, pages 283–288. ACM, 2011.

[19] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N. Calheiros, Soumya K. Ghosh, and Rajkumar Buyya. Fog Computing: Principles, Architectures, and Applications. *arXiv:1601.02752 [cs]*, January 2016. arXiv: 1601.02752.

[20] C. Dsouza, G. J. Ahn, and M. Taguinod. Policy-driven security management for fog computing: Preliminary framework and a case study. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 16–23, August 2014.

[21] Jochen Eisl, Albert Rafetseder, and Kurt Tutschku. Service architectures for the future converged internet: Specific challenges and possible solutions for mobile broad-band traffic management. *Advances in Next Generation Services and Service Architectures*, October 2010.

[22] Philipp M Eittenberger, Marcel Großmann, and Udo R Krieger. Doubtless in seattle: Exploring the internet delay space. In *Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on*, pages 149–155. IEEE, 2012.

[23] V. Gazis, A. Leonardi, K. Mathioudakis, K. Sasloglou, P. Kikiras, and R. Sudhaakar. Components of fog computing in an industrial internet of things context. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*, pages 1–6, June 2015.

[24] Google cloud iot core. https://cloud.google.com/iot-core/, last access: Jun 2017.

[25] Sara Hooshangi, Richard Weiss, and Justin Cappos. Can the security mindset make students better testers? In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 404–409. ACM, 2015.

[26] B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe. Evaluation of Docker as Edge computing platform. In *2015 IEEE Conference on Open Systems (ICOS)*, pages 130–135, August 2015.

[27] Tao Li, Albert Rafetseder, Rodrigo Fonseca, and Justin Cappos. Fence: protecting device availability with uniform resource control. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, pages 177–191. USENIX Association, 2015.

[28] Redowan Mahmud and Rajkumar Buyya. Fog Computing: A Taxonomy, Survey and Future Directions. *arXiv:1611.05539 [cs]*, November 2016. arXiv: 1611.05539.

[29] X. Masip-Bruin, E. Marín-Tordera, G. Tashakor, A. Jukan, and G. J. Ren. Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems. *IEEE Wireless Communications*, 23(5):120–128, October 2016.

[30] Paul Müller, Dennis Schwerdel, and Justin Cappos. Tomato a virtual research environment for large scale distributed systems research. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, 37(1):23–32, 2014.

[31] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[32] Albert Rafetseder, Florian Metzger, Lukas Pühringer, Kurt Tutschku, Yanyan Zhuang, and Justin Cappos. Sensorium–a generic sensor framework. *Praxis der Informationsverarbeitung und Kommunikation*, 36(1):46–46, 2013.

[33] M. Reininger, S. Miller, Y. Zhuang, and J. Cappos. A first look at vehicle data collection via smartphone sensors. In *2015 IEEE Sensors Applications Symposium (SAS)*, pages 1–6, April 2015.

[34] S. Tayeb, S. Latifi, and Y. Kim. A survey on iot communication and computation frameworks: An industrial perspective. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–6, Jan 2017.

[35] ToMaTo Testbed. http://tomato-lab.org/.

[36] Stephen Tredger, Yanyan Zhuang, Chris Matthews Jesse Short-Gershman, Yvonne Coady, and Rick McGeer. Building green systems with green students: An educational experiment with geni infrastructure. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pages 29–36. IEEE, 2013.

[37] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane. Software defined networking-based vehicular Adhoc Network with Fog Computing. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1202–1207, May 2015.

[38] Kurt Tutschku, Albert Rafetseder, Jochen Eisl, and Werner Wiedermann. Towards sustained multi media experience in the future mobile internet. In *2010 14th International Conference on Intelligence in Next Generation Networks (ICIN)*, Berlin, Germany, October 2010.

[39] Scott A. Wallace, Monzur Muhammad, Jens Mache, and Justin Cappos. Hands-on internet with seattle and computers from across the globe. *J. Comput. Sci. Coll.*, 27(1):137–142, October 2011.

[40] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog Computing: Platform and Applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78, November 2015.

[41] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42, New York, NY, USA, 2015. ACM.

[42] Yanyan Zhuang, Leonard Law, Albert Rafetseder, Lai Wang, Ivan Beschastnikh, and Justin Cappos. Sensibility testbed: An internet-wide cloud platform for programmable exploration of mobile devices. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 139–140. IEEE, 2014.

[43] Yanyan Zhuang, Chris Matthews, Stephen Tredger, Steven Ness, Jesse Short-Gershman, Li Ji, Niko Rebenich, Andrew French, Josh Erickson, Kyliah Clarkson, et al. Taking a walk on the wild side: teaching cloud computing on distributed research testbeds. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 535–540. ACM, 2014.

[44] Yanyan Zhuang, Albert Rafetseder, and Justin Cappos. Privacy-preserving experimentation with sensibility testbed. *;login:: the magazine of USENIX*, 40(4):18–21, 2015.

[45] Yanyan Zhuang, Stephen Tredger, Chris Matthews, Rick McGeer, and Yvonne Coady. Distributed systems in the wild: The theoretical foundations and experimental perspectives. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2012 Seventh International Conference on*, pages 87–94. IEEE, 2012.