Shaping Smart City Systems

Rodney Holman
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
holma198@morris.umn.edu

ABSTRACT

Increasing urban populations and higher standards of living demand more city services and more prompt responses of these services. A smart city could potentially reach this demand, and in recent years, smart city technologies have become more and more common. Smart cities are comprised of many technology systems that communicate with each other and aid in the task of day-to-day municipal functions. This concept is still relatively new and the term "smart city" is not standardized to one definition. To further complicate things, many problems can arise if implementation of smart city systems is done improperly. The purpose of this paper is to highlight the benefits of a smart city with real world examples, the functionality of a smart city system, and finally a way to avoid potential risks from a smart city system.

Keywords

Smart City, SOXFire, Internet of Things

1. INTRODUCTION

Traffic management, air quality control, and garbage collection are just a few of the typical services a city provides. Without these services, cities and quality of life can suffer drastically, and unfortunately this is often the case in far too many urban areas. Over the years, the blanket term "smart city" has been used to describe a city which has adopted a network of information technologies to be able to react to and mitigate some of these daily urban issues.

To illustrate the demands of a smart city, it would be wise to first examine the benefits of a smart city. An example of usefulness can be seen in India's "Smart Cities Mission". This program aims to select certain cities to ultimately implement smart city systems. The goal of this smart city program is to improve walk-ability, public transit, city governance accessibility to citizens, and infrastructure among other goals. In the case of India, each of these goals are achieved through programs within the smart city initiative [2]. A specific example of a problem being solved with a smart city system would be digital identity. Rivera et al. [4] utilizes a Blockchain network, which in essence is a secure distributed and decentralized database network. This system simplifies city services that require citizen interaction such as payment for certain services, or it can be used for census information. Another example demonstrated by Yonezawa et al. [8] would be vehicle-mounted sensors to gather city data, such as pollutants, to store data over a network from which results can be gathered and responded to appropriately.

This paper provides a look into the fundamental processes of a smart city system, the network behind a smart city system, and finally conflicts that can happen between the networks in the system as well as ways of mitigating these conflicts.

The network is fundamental to the functionality of all the parts in a smart city. The network interface, *SOXFire*, that this paper examines in Section 3 is implemented by [8]. This network aims at being scalable for any city's size and demands while being robust and specific enough for each of the city's services. For instance, the network would have to process the data that sensors produce from a pedestrian service and the data a sensor collects on pollution levels from a pollution service. This data is later visualized with visualization tools as depicted in Section 4.

Lastly, conflicts between networks are discussed in Section 5. In addition, a solution called *CityGuard* is presented by Meiyi Ma et al. [3]. This solution utilizes an algorithm that is run against a simulation of services within New York City, NY. The algorithm attempts to resolve the conflicts in every network to maintain safety within the city.

2. BACKGROUND

To fully understand smart city systems, it is first important to understand the underlying basic processes of a smart city. The first portion of this background gives an overview of these processes. This is followed up by information regarding the sensors that make up a smart city.

2.1 Smart City Process

Each smart city strives in some way to meet the following goals: data collection, data analysis, and reaction to the data. While services may vary by municipality, the process remains the same. This process can be viewed on a macro scale and in a constant loop as depicted in Figure 1 [5]. Ideally this loop represents every system in a smart city, as opposed to each system individually. In this loop, the city provides data through a network of various sensors. This data is then analyzed through models and analytical tools which is acted upon by stakeholders and citizens of the city. This affects the city in different ways which creates new data for the network, and the loop continues in this cy-

UMM CSci Senior Seminar Conference, April 2019 Morris, MN.

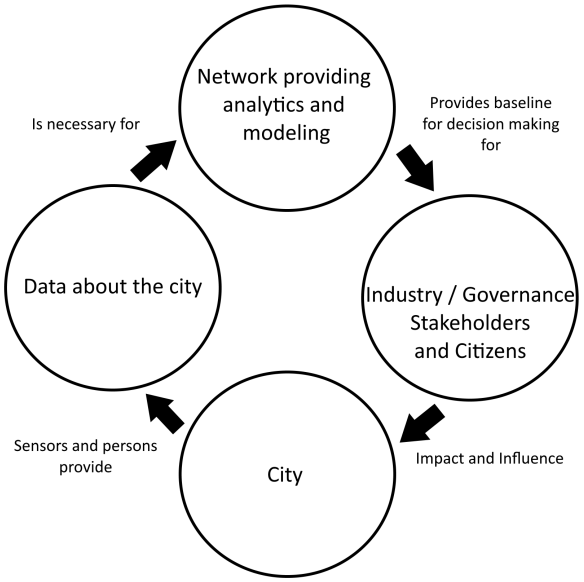

Figure 1: A Smart City Loop [5].

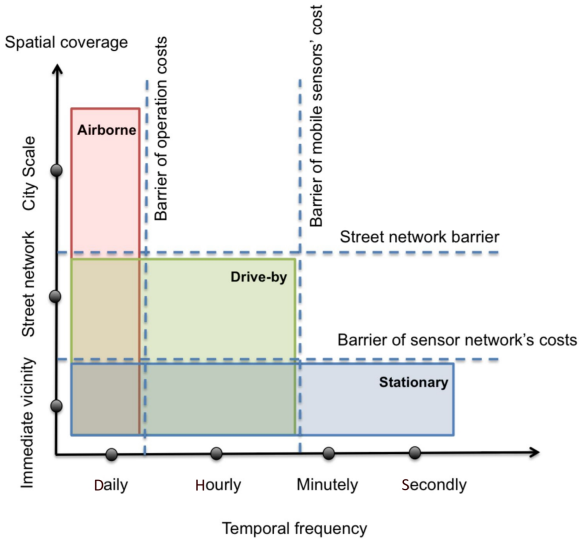

Figure 2: Comparing spatial (distance) and temporal (time) benefits of sensor types [1].

cle. Another approach of classifying smart city goals derives from the "3-C concept" as described by Kumar et al. [2]. This concept is simply an acronym meaning "Convenience, Competence, and Cleverness". These terms highlight what attributes describe an ideal network of smart city systems and how each smart city service should function.

2.2 Sensor Types

A smart city is a network of sensors that report data through various means. In order to understand how that data is reported through *SoxFire*, it is first important to know what kinds of sensors and data need to be accounted for. Each sensor and data type require different approaches for monitoring and interpretation. The benefits and disadvantages of each sensor based on its application are pictured in Figure 2. The Figure depicts spatial coverage, which is the amount of area a sensor can reasonably cover, and temporal frequency which is the how frequently data can be collected for a given area. An example would be a satellite (airborne) having the advantage of being able to record data on a large scale but also having the disadvantage of being able to do so infrequently due to computational and cost restraints as well as the mobility of the sensor changing from a given area. There are many different types of sensors for collecting information in a city. The three primary types of sensors are stationary sensors, *Drive-By* or vehicle-mounted sensors, and crowd sensing sensors.

2.2.1 *Fixed Sensors*

Fixed sensors, also known as stationary sensors, are the most common devices for gathering information in a smart city application, particularly for environmental sensing. This form of collecting information is usually accurate but limits the area that can be covered due to its static location. These sensors are typically more expensive than using vehicle-mounted sensors due to a greater number required for the same area, installation costs, and required maintenance at each stationary sensor location. [1]

2.2.2 *Vehicle-Mounted Sensors*

An alternative to fixed sensors are vehicle-mounted sensors. These "drive by" sensors can provide information at a much greater scale than fixed sensors. These sensors are also useful for environmental data, and some specific applications include gas leak and pothole detection as well as pollution monitoring [1]. Some specific sensor types used on vehicles include but are not limited to UV, temperature, and nitrogen dioxide/nitric oxide pollution (NOx) sensors.

2.2.3 *Crowd Sensing*

Certain types of data require more detail than a quantitative data sensor can provide. Within this category of data collection are two subcategories provided by Yonezawa et al. [8] : participatory and expert crowd sensing. Participatory crowd sensing involves regular citizens voluntarily reporting something about the city. An example would be a large pothole being reported by a concerned citizen. Alternatively expert crowd sensing would involve city officials and employees gathering and reporting data specific to their own daily tasks. For both types of data collection, the typical data being collected is an image and/or a form of some kind sent via a smartphone app, though quantitative data can also be provided [8].

3. THE NETWORK

The network is vital for storing the information gathered from sensors in a database such that actions can be taken from the data. While there are no restrictions to a particular network, there are a few desirable qualities of a smart city network that should be accounted for.

3.1 Network Requirements

Perhaps the biggest issue a smart city network is confronted with is the different types of sensors. The smart city network must be able to record data from all these sensor types regardless of differences in data types and request frequency. Further complicating the problem space, the system

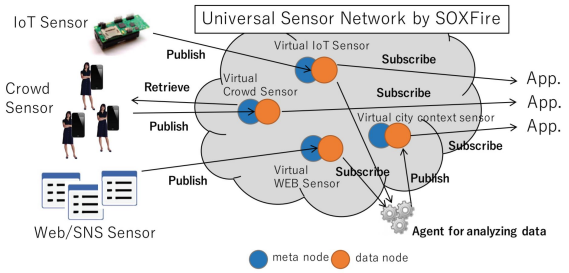

Figure 3: Virtual Sensors in SOXFire [8].

must also be properly extendable and scalable to account for numerous different city departments at a city scale. *Federation* is how this is achievable. Federation, in a general sense, is a group of networks agreeing upon operation standards collectively [6]. This is particularly important with user identity and authentication for a smart city application. In addition, federation also improves extendability of the networks in the system. Federation will be further explored in Section 3.2.1. Lastly, security poses another issue for data collection, which needs to also be balanced with the accessibility and usability required by users participating in crowd sensing [8].

3.2 SoxFire

A working implementation of a smart city network interface introduced by Yonezawa et al. [8] called *SoxFire* addresses these requirements. This network was used in Fujisawa, Japan and field tested on both vehicle and crowd sensors. SOXFire is open source and built upon an XMPP protocol.

3.2.1 XMPP

The XMPP protocol is used for several applications, but primarily for online chat applications such as "WhatsApp". The term *SOX* in SOXFire is derived from the term Sensor-Over-XMPP. XMPP is particularly useful because it comes packaged with many of the useful features and requirements built in, such as authentication and encryption, and the ability to easily add custom functionality as was done with SOXFire. Lastly, federation in XMPP includes several properties useful for SOXFire. XMPP federation includes two primary features: *decentralization* and *gateways*. The protocol is decentralized by providing each network user with a unique id called a *Jabber ID* or *JID*. Using the JID, the server can be run on any domain. Gateways are utilized by XMPP to allow a single client to access other networks even if they are not XMPP. In addition to this, users can register with the gateway to auto-authenticate. [7] To visualize a gateway, it may be helpful to think of a bouncer at a prestigious club. They will let you in, no questions asked, if they already know who you are. These features are why XMPP federation is fundamental for extendability and scalability with many separate sensor networks.

3.2.2 Virtual Sensors

SOXFire is composed of "virtual" sensors which are useful for scalability of the differing sensor types. These virtual sensors are composed of two nodes which include data information and meta information. The data would simply be specific to *what* a certain sensor recorded whereas the meta node would contain information about *which* sensor published the information. For example, a temperature sensor would report that it is 75 degrees outside as data, and report that it is located on top of a building on 7th Street as metadata.

Another component of these virtual sensors would be publish and subscribe events, alternatively known as "PubSub" events. Each virtual sensor corresponds to a real sensor in the city. Whenever a sensor publishes information, the virtual sensor retrieves both the data and metadata of the sensor and the application(s) in which the data is used subscribes to these events.

In the case of SOXFire, crowd sensing data is published to a virtual sensor by groups of individuals whom voluntarily provide this information. Another case specific to SOXFire involves web scraping, which utilizes virtual sensors on relevant web pages. This is useful for monitoring information that is outside of a city's sensor network but may still be useful for the city. Figure 3 displays how the various methods of collecting information in the city are handled by SOXFire's virtual sensors and PubSub events.

3.2.3 The Backend

The server implementation of SOXFire is built upon OpenFire, which is designed for XMPP protocols that use Java. The primary difference of SOXFire's implementation is that it allows subscription and federation simultaneously. This is necessary for sensors to seamlessly communicate with sensor networks, and each sensor network to have subscriptions to the virtual sensors amongst the entire city's physical sensors. In addition user connections are monitored to keep track of metadata. This metadata is used to keep track of the location and the service that the data is originating from so the information can be organized and categorized by origin.

The server API is straightforward and hides the two different node types from the user. It is designed to be easy to use and focus primarily on the subscription and publishing events from the virtual sensors.

3.2.4 Results

SOXFire's implementation in Fujisawa, Japan is still experimental with field trials being run rather than being a finished product ready for any city to adopt. Regardless, it still provides a compelling proof of concept that successfully handles data of various kinds (air pollution and crowd sensing). In addition, the network has a visualization dashboard application as discussed in Section 4.

4. DATA INTERPRETATION

Among the most important aspects of a smart city is interpreting what data is recorded and appropriately acting upon it. Many smart city systems provide tools to help visualize and locate problems within a city.

In the SOXFire implementation, once data is gathered it is represented through a "dashboard" view as seen in Figure 4. This view displays text, image and map data that are gathered in real time from the sensor network. The view is customizable with widgets that can be moved and added specific to the user. The Figure depicts various information collected by services in Fujisawa, Japan, including a city monitoring camera, participatory crowd sensing widget, and a weather widget.

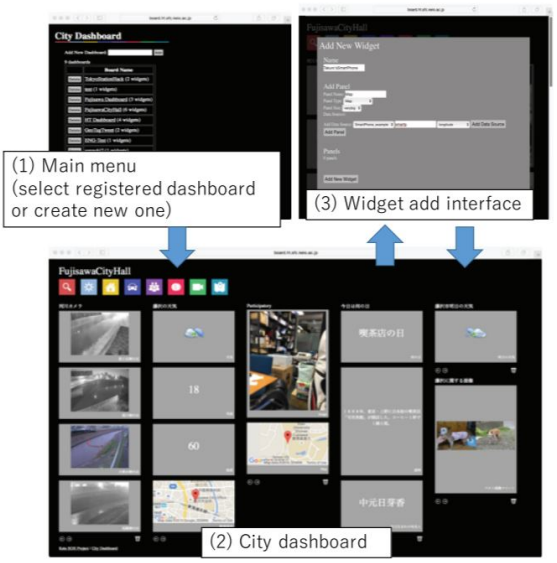

Figure 4: SoxFire Dashboard [8].

5. SENSOR SYSTEM CONFLICTS

Unfortunately, simply meeting the network requirements of a sensor system and appropriately gathering senor data is not enough to have a safe and efficient smart city. Issues can occur within the greater network of sensors in which certain smart city systems may violate other smart city system goals without proper communication between the systems. This is often the case due to different safety requirements and end goals of each system and different groups developing and maintaining these systems [3]. Each service in the city is likely to have its own sensor system with its own network, which can further increase the likelihood of problems arising. A service, in this case, would be any entity in the city whose objective is to accomplish a set of tasks or maintain certain goals. Meiyi Ma et al. [3] describe the various types of conflicts between smart city systems and a feedback loop based algorithm which addresses these issues.

5.1 Conflict Types

The two primary types of conflicts defined in [3] are device and environmental.

Device conflicts involve a single device being given two commands at once. Opposite actions as well as similar but different actions fall under this category. An example would be if a traffic service requests to turn a traffic light green and a pedestrian service requests that same light to be red. Specifically, this is an opposite device conflict, because opposite actions are requested on the same device.

Environmental conflicts are those that involve multiple devices and systems. A good example of a smart city system conflict would be between an emergency services system and a traffic system. If a fire department requires a given street and traffic has been routed to that street, there is a conflict of systems. Conflicts like these can be mitigated first by defining them. This conflict can be defined as any action taken on one system that violates an action or safety threshold of another.

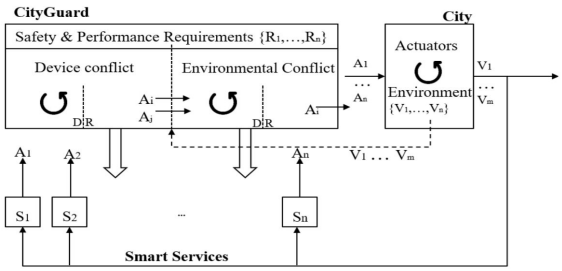

Figure 5: CityGuard Overview [3].

5.2 CityGuard

CityGuard addresses both conflict types in a feedback loop: a situation in which the output of the loop continuously becomes the input of the loop. In the case of CityGuard, each system has actuators that automate actions of those systems on a macro scale. These actuators are activated based on the output of the city information gathered by the sensor networks. Whenever the actuators launch actions, the results of those actions are continuously fed in. CityGuard interjects conflict mitigation between city system actions and the actuators as seen in Figure 5; this conflict mitigation attempts to make service actions in the city safer.

In Figure 5, the services $\{S_1 \ldots S_n\}$ all have actions $\{A_1 \ldots A_n\}$ to be analyzed in CityGuard. The state of the city is simulated with each action; because of this, each action has its own simulation state associated with it $\{V_1 \ldots V_n\}$. These are all necessary for CityGuard to find and resolve device and environmental conflicts. After CityGuard runs, the resolved actions are given to the actuators so that they can be activated in the city.

CityGuard operates on a number of "rules". These rules are defined by the city in which it is implemented. A rule could be a noise threshold for a residential neighborhood. CityGuard could not violate that noise threshold regardless of the circumstances. Because CityGuard cannot violate these rules, it may not always mitigate a conflict, but it can reduce the severity of the conflict. These rules are known by CityGuard as requirements. These requirements are visible in Figure 5 as $\{R_1 \ldots R_n\}$.

Much like SOXFire, CityGuard utilizes metadata of individual sensor devices to properly identify where and which department requested an action. In addition to this information, the duration and prior conditions of an action are interpreted.

5.2.1 Conflict Detection and Resolution

In order to understand the algorithm behind CityGuard, it is important to understand the sub components in conflict detection and resolution. Figure 6 displays the algorithm as a diagram with additional safety and performance requirements, states of the city, and service actions. This diagram also displays the various sub components within CityGuard.

The first sub component is CityGuardSUMO, Simulation of Urban MObility, which utilizes several simulated states of the city to monitor and predict secondary effects of traffic systems, such as increased congestion from a change in a traffic light at an intersection. The secondary effects would be classified as possible environmental conflicts, much like the example with the fire department and traffic system.

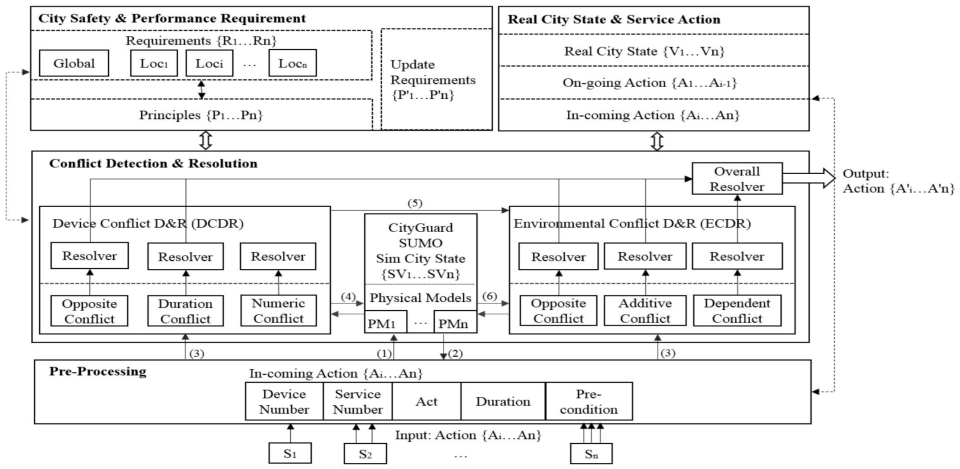

Figure 6: CityGuard Algorithm [3].

Device Conflict Detection and Resolution is another sub component which is used to monitor devices for conflicts of any kind. Conflicts are organized into three groups: Opposite, Duration, and Numeric. Each device is checked for a conflict in any of these conflict groups, and if one is found it is resolved with a corresponding resolver to that conflict group. An example could be autonomous vehicles around the city being set to 50 MPH by one service, and 60 MPH by another service. A device conflict has occurred, specifically an numeric device conflict, and it will be resolved in the corresponding resolver of that conflict type.

Environmental Conflict Detection and Resolution takes the returned environmental conflict values from SUMO and classifies them into categories (single, opposite, additive, and dependent conflicts) similarly to the Device Conflict sub component. The primary difference from SUMO is that this sub component checks for additive affects, meaning that all the possible conflicts are checked together for classification in this conflict category. If, for instance, a traffic service routes traffic onto a road and a waste management service directs a fleet of garbage trucks onto that same road an additive violation has occurred on that road for congestion, even if no services by themselves have violated that congestion threshold. Dependent environmental conflicts are also checked. A dependent conflict could be a traffic service routing traffic while an accident service is still mitigating a traffic accident and needs the road cleared. The traffic service is dependent on the accident service to complete.

The Overall Resolver is the last check before actions are made in the city. Actions deemed as safe by both of the two conflict resolvers are considered safe here. Actions deemed unsafe by the Environmental Conflict Resolver are rejected here. Actions deemed unsafe by the Device Conflict Resolver are rechecked for other possible actions that will not violate any safety thresholds. If no alternative action is found, it is rejected. If an action is found it is then considered safe and the action is applied.

5.2.2 Algorithm

CityGuard utilizes an algorithm with three main steps: Pre-processing, Device Conflict, and Environmental Conflict (see Figure 6).

Pre-processing: involves a set of actions $\{A_1 \ldots A_n\}$, a set of simulation states $\{SV_1 \ldots SV_n\}$, and a set of city states $\{V_1 \ldots V_n\}$ that the simulation state is set to. For each action A in the action set $\{A_1 \ldots A_n\}$, the current set of simulation states are set to the result of CityGuardSUMO with the arguments $\{A_1 \ldots A_n\}$ and $\{SV_1 \ldots SV_n\}$. This result and the requirement of the action is determined to be safe or not, and if it is not safe it is resolved in the resolver.

Device Conflict: The Action set $\{A_1 \ldots A_n\}$ is passed into the device checker. If the result of the device checker yields no new conflict, this is passed to the Environmental Conflict portion of the algorithm. If a device conflict is found by the device checker, each device conflict case is classified and resolved in a resolver for that type of conflict.

Environmental Conflict: CityGuardSUMO is run with Pre-processed action set $\{A'_1 \ldots A'_n\}$ and the city simulation states $\{SV_1 \ldots SV_n\}$. This result is stored as $\{SV'_1 \ldots SV'_n\}$. $\{SV'_1 \ldots SV'_n\}$ and the city requirements $\{R_1 \ldots R_z\}$, z in this case being the number of requirements, are checked with the Environmental Checker to be classified as a conflict or not and of what type of conflict. Each action in the action set $\{A'_1 \ldots A'_n\}$ is resolved in the corresponding resolver of its category and finally passed to the Overall Resolver.

5.2.3 Results

CityGuard was evaluated in a simulation of Manhattan, NY extended from SUMO data. Various city services are introduced in this simulation, ranging in priority from a traffic congestion service to an emergency response service. CityGuard drastically improved safety metrics such as halving the wait time of emergency vehicles in some instances. Wait time is measured by the length of time a vehicle waits in a lane at an intersection. Traffic collisions were also successfully mitigated when running CityGuard. There are compromises, however, that are necessary to make these gains. In the simulation, normal vehicle wait time increased from 98.5 seconds to 100.2. It is important to note that this is still lower than the 121.82 second wait without any services. The improvements that CityGuard offers become more pronounced when more services are running. If 5 services are running, CityGuard reduces CO emission by 51.3%. That

reduction increases to 73.7% when 10 services are introduced.

The simulation also revealed information about the services themselves and their interactions. The accident service had the largest spatial impact while the noise mitigation service had the least. Device conflicts were prominent throughout the simulation; the congestion and pedestrian services had conflicts with one another around 48.3% of the time. Looking at environmental conflicts, it was found that running 7 services or more significantly increases the chance of conflicts. Environmental conflicts were found to occur 2-4 times more frequently than device conflicts [3].

6. CONCLUSIONS

Smart city systems may not fix every urban problem completely, however, the proper network and conflict mitigation systems can yield some compelling results. This paper has examined the network requirements of a smart city necessary for intracity communication between devices. While SOXFire specifically is one of many possible solutions, it does satisfy the network requirements for a smart city network and successfully accomplishes its task for retrieving and publishing city system actions. Further work still needs to be done to fully explore the network's capabilities and to add to the data analysis portion of the system to further automate the systems within that smart city.

CityGuard clearly shows positive results in a wide range of categories, but most importantly in safety. A conflict mitigation system like CityGuard not only shows exceptional promise, but it appears to become necessary as more city systems are added to the smart sensor network. In addition to this, the simulation environment that CityGuard is tested in introduces automated responses to sensor data, which can mitigate the shortcomings of SOXFire. CityGuard itself, however, will need further testing in a real world environment before it can be proven viable for a real smart city.

Acknowledgments

Thanks to KK Lamberty, Ian Buck, and Elena Machkasova for their advice and feedback.

7. REFERENCES

[1] A. Anjomshoaa, S. Mora, P. Schmitt, and C. Ratti. Challenges of drive-by IoT sensing for smart cities: City scanner case study. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, UbiComp '18, pages 1112–1120, New York, NY, USA, 2018. ACM.

[2] N. M. Kumar, S. Goel, and P. K. Mallick. Smart cities in India: Features, policies, current status, and challenges. In *2018 Technologies for Smart-City Energy Security and Power (ICSESP)*, pages 1–4, March 2018.

[3] M. Ma, S. M. Preum, and J. A. Stankovic. Cityguard: A watchdog for safety-aware conflict detection in smart cities. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pages 259–270, New York, NY, USA, 2017. ACM.

[4] R. Rivera, J. G. Robledo, V. M. Larios, and J. M. Avalos. How digital identity on blockchain can contribute in a smart city environment. In *2017 International Smart Cities Conference (ISC2)*, pages 1–4, Sep. 2017.

[5] J. M. Schleicher, M. Vögler, C. Inzinger, and S. Dustdar. Towards the internet of cities: A research roadmap for next-generation smart cities. In *Proceedings of the ACM First International Workshop on Understanding the City with Urban Informatics*, UCUI '15, pages 3–6, New York, NY, USA, 2015. ACM.

[6] Wikipedia. Federation (information technology) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Federation%20(information%20technology)oldid=866156138, 2019. [Online; accessed 30-March-2019].

[7] Wikipedia. XMPP — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=XMPPoldid=889703389, 2019. [Online; accessed 30-March-2019].

[8] T. Yonezawa, T. Ito, J. Nakazawa, and H. Tokuda. Soxfire: A universal sensor network system for sharing social big sensor data in smart cities. In *Proceedings of the 2nd International Workshop on Smart Cities*, SmartCities '16, pages 2:1–2:6, New York, NY, USA, 2016. ACM.