CityGuard: A Watchdog for Safety-Aware Conflict Detection in Smart Cities

Meiyi Ma
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903
meiyi@virginia.edu

Sarah Masud Preum
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903
preum@virginia.edu

John A. Stankovic
Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903
stankovic@virginia.edu

ABSTRACT

Nowadays, increasing number of smart services are being developed and deployed in cities around the world. IoT platforms have emerged to integrate smart city services and city resources, and thus improve city performance in the domains of transportation, emergency, environment, public safety, etc. Despite the increasing intelligence of smart services and the sophistication of platforms, the safety issues in smart cities are not addressed adequately, especially the safety issues arising from the integration of smart services. Therefore, CityGuard, a safety-aware watchdog architecture is developed. To the best of our knowledge, it is the first architecture that detects and resolves conflicts among actions of different services considering both safety and performance requirements. Prior to developing CityGuard, safety and performance requirements and a spectrum of conflicts are specified. Sophisticated models are used to analyze secondary effects, and detect device and environmental conflicts. A simulation based on New York City is used for the evaluation. The results show that CityGuard (i) identifies unsafe actions and thus helps to prevent the city from safety hazards, (ii) detects and resolves two major types of conflicts, i.e., device and environmental conflicts, and (iii) improves the overall city performance.

CCS CONCEPTS

•Computing methodologies → Modeling and simulation; • Computer systems organization → Embedded and cyber-physical systems; •Applied computing → *Service-oriented architectures;*

KEYWORDS

City Safety, Conflict Detection, City Simulation, Smart City

ACM Reference format:
Meiyi Ma, Sarah Masud Preum, and John A. Stankovic. 2017. CityGuard: A Watchdog for Safety-Aware Conflict Detection in Smart Cities. In *Proceedings of The 2th ACM/IEEE International Conference on Internet-of-Things Design and Implementation, Pittsburgh, PA USA, April 2017 (IoTDI 2017),* 12 pages.
DOI: http://dx.doi.org/10.1145/3054977.3054989

IoTDI 2017, Pittsburgh, PA USA

DOI: http://dx.doi.org/10.1145/3054977.3054989

1 INTRODUCTION

With the increasing prevalence and development of smart cities, the intelligence and complexity of smart services and platforms have been growing rapidly. Various smart services in different domains, such as, transportation, environment, energy, and emergency management have been separately deployed to improve city operation and human experiences. For example, a smart energy service [17] distributes power optimally and saves idle energy, and a smart taxi service [13] dispatches taxis to minimize passenger wait time. To further increase smart city capabilities, there is a need for integration of multiple smart devices and services. Smart city IoT platforms, such as IBM Watson IoT [3], ORACLE [5], KM4City [8], and others are built to integrate smart services working under the same system, making the communication and data sharing among them possible and convenient.

However, to date, most of existing smart cities do not consider safety control in the context of integrated smart services, especially in terms of conflicts among them. Not enough attention is paid to the potential safety conflicts caused by conflicting actions taken by different smart services. For example, assume a smart traffic service redirects vehicles from a highway to another road near a residential area to relieve traffic congestion. Now a higher concentration of CO will be released in the residential area. Assume that the smart traffic service is aware of this problem and regulates traffic accordingly. Now assume, there is a chemical factory nearby the residential area which also releases CO gas but is monitored to guarantee that CO release is under safe limits. However, with this new action of diverting traffic, the total concentration of CO now exceeds safe limits. This results in unsafe air quality, and affects the health of people living in this area. Also, although the two services maintain the safety requirement (i.e., keeping CO release below safety threshold) individually, their concurrent operations violate the safety requirements cumulatively. Thus, it causes a conflict. Safety maintenance in a smart city is extremely important and challenging.

Even though some smart services have individual sophisticated decision-making processes that consider safety requirements, there still exist inevitable conflicts between smart services in smart cities because of the following reasons:

- Most of the existing smart services are developed and used independently by different stakeholders including governments, commercial enterprises, and individuals. Also, the individual services usually have varied levels of sophistication and attention on safety requirements, which may be unaware of or inconsistent with each other.

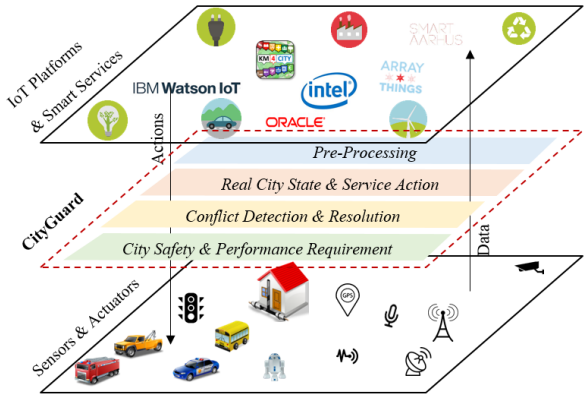

Figure 1: CityGuard in the Smart City

- In most cases, the safety requirements from services only consider the primary effects (e.g., minimize congestion) while ignoring secondary effects on the environment (e.g., air pollution), thus violating safety requirements.
- The *safe* actions defined by smart services may not always be safe for a smart city when considering (i) the effects of those actions over a time interval, an area and (ii) the actions taken by other services.
- Devices and services might be updated over time without notifying other services. Even though services may have pre-defined safety requirements and rules to avoid conflicts, new conflicts may arise as smart services evolve over the long term.

Hence, a safety-aware watchdog architecture, called CityGuard, is created to detect and resolve the conflicts from multiple smart services in a smart city (see Figure 1). Generally, the smart city is constructed of two main parts, (i) an infrastructure layer with sensors and actuators and (ii) a software layer consisting of IoT platforms and individual smart services. CityGuard is designed to be a middle layer with sophisticated safety requirements that is embedded between the infrastructure layer and services themselves. CityGuard intercepts the actions from smart services, detects whether potential conflicts exist ahead of time and provides resolution whenever possible.

1.1 Challenges

The key challenges in building a safety watchdog for smart cities arise from the following three aspects.

1.1.1 Defining Safety Requirements in Smart Cities. In order to keep a safe environment, rigorous and consistent specifications of the safety requirements for both a smart city and its smart services are important. However, it is difficult to integrate all the safety requirements from all services, because they are defined under their own contexts with different granularity. Furthermore, another challenge is how to define safety requirements in the watchdog considering all the competing objectives of the smart city.

1.1.2 Detecting Conflicts of Smart Services. Detecting and resolving the conflicts of smart city services are both significant and difficult for the following reasons.

- The effects of an action are hard to predict in a complex city environment, especially the secondary effects, which are also likely to violate city safety requirements.
- In addition to direct conflicts, there are indirect conflicts. Indirect conflicts result from concurrent operations of multiple actions, each of which is safe individually, but becomes conflicting when performed simultaneously.
- Effects of actions last over varying time intervals and areas and conflicts may arise during those overlapping intervals.

1.1.3 Implementation. Due to the extremely complex functionalities of a smart city, it is challenging to integrate all the smart services and implement a watchdog architecture like CityGuard. The implementation must be able to predict the effects of actions with reasonable accuracy. Presumably, a combination of mathematical models, environmental models, human behavioral models, and heavy use of simulators will all be needed to detect and resolve conflicts.

1.2 Contributions

To the best of our knowledge, CityGuard is the first work to build a safety-aware watchdog for detecting and resolving conflicts between services to address city safety requirements with three major underlying contributions.

1.2.1 Specification and Definition of Safety Requirements and Conflicts. CityGuard specifies a set of safety requirements and identifies a broad spectrum of conflicts in smart cities that are aware of the objectives of smart services.

1.2.2 Detection of Conflicts. CityGuard detects different types of conflicts by intercepting actions ahead of time, analyzing the details of the actions, and then running simulations to predict potential conflicts within a temporal and a spatial range. This paper focuses on conflict detection, especially for environmental conflicts. In addition, conflict resolution with priority-based rules are provided.

1.2.3 Simulation and Evaluation on the Smart City. Based on the real data collected from New York City, a smart city simulator, namely CityGuard SUMO, is built with an emphasis on the domains of transportation, environment, and emergency services to demonstrate potential conflicts in a real city. Simulation is done in a controlled setting. Ten types of smart services are implemented and then distributed to 20 major locations in the simulated Manhattan, (i.e. 200 smart services are running in parallel). Each component of CityGuard is evaluated by comparing results from the following 3 scenarios: (i) city without smart services, (ii) city with smart services, and (iii) city with smart services and CityGuard. The results show that CityGuard helps smart services running under safety and performance requirements by identifying and preventing unsafe actions as well as detecting and resolving conflicts.

In regards to city safety, CityGuard prevents up to 20 traffic collisions caused by the 10 services within an hour interval, and saves up to 101% waiting time for emergency vehicles. In terms

of city performance, CityGuard increases air quality by 73.7% and decreases vehicle waiting time by 35% over a baseline system (e.g., city operating without CityGuard).

2 CITY SAFETY AND CONFLICT

In order to fully understand CityGuard and its scope, in this section, we define terms related to the safety requirements, the effects of actions from services, and the spectrum of conflicts that are being considered.

2.1 Safety Requirements

2.1.1 Coverage. City safety generally involves safety for the environment and humans. **Environmental Safety** includes maintaining the quality of air, water, noise, and weather and safety of property. For example, actions taken by a service should not result in air, water, or noise pollution, or street lights should be kept on at night in crime-prone areas. **Human Safety** refers to protecting citizens from any dangerous or unhealthy situation (e.g., road accidents).

Consequently, the overall *Smart System Safety* means that all the smart services running in a city must neither bring danger to the environment or citizens, nor conflict with each other. For example, autonomous vehicles are not allowed to hit pedestrians or cause environmental damage. Also, a smart traffic service and a smart emergency service should not try to turn the same traffic light to green and red simultaneously.

2.1.2 Context. In the complex and dynamic setting of real time smart cities, safety and performance requirements need to be aware of context, i.e., they need to accommodate special circumstances, because rigid/static safety requirements can result in unwarranted / catastrophic consequences. For example, assume a transportation domain service sets the highest traffic capacity of a street, *street A* to 100 to avoid congestion (i.e., a performance metric) although the street may accommodate more vehicles for a short time. However, when there is a fire on a nearby street, *street B* vehicles may have to evacuate through *street A* (i.e., for safety). In this circumstance, the performance requirement of maintaining traffic capacity in *street A* should be aware of the safety requirement of *street B*. Thus the safety and performance requirements should be context aware.

2.1.3 Emphasis. As another complication, range and frequency of temporal and spatial entities should also be considered when specifying unsafe situations. For example, the traffic congestion lasting for 5 min might not be considered as an unsafe situation, but the one lasting for 1 hour is unsafe.

2.2 Effects of Actions from Services

2.2.1 Primary and Secondary Effects. After an action is taken, it has a series of effects on the city. A primary effect relates to the main purpose of the action. For example, to control the noise level in a school area, the noise control service does not allow trucks to go through the school area during the day and redirects them to a nearby residential area. The primary effect of this action is the reduction of the noise level in the school area during day time. However, this action may result in one or more secondary effects. For example, in this case, the traffic volume of the nearby residential area may increase as some traffic is redirected from the school area. Such secondary effects may have a serious influence on the city environment, which may violate the city safety requirements.

Table 1: Examples of conflicts of services in smart cities

Category	Type	Example
Device	Opposite	Pedestrian service turn a traffic signal to green, while Traffic Congestion Service turn the same traffic signal to red.
Device	Numeric	Traffic Service set the speed of autonomous vehicles to be 70 mph; Safety Service set the speed of them to be 60 mph.
Device	Duration	Emergency service keeps the traffic lights *green* for 10 minutes to allow ambulances to move faster while traffic congestion service needs it to turn *red* every 2 minutes.
Environment	Single	Air quality control service redirect the traffic to reduce air pollution, but cause serious traffic congestion on the other road, level of which exceeds safety requirement.
Environment	Opposite	While emergency service tries to evacuate an area, traffic congestion service directs more vehicles there.
Environment	Additive	Event service caused a certain level of noise below threshold, emergency service caused a level of noise below threshold, but the additive level is above threshold.
Environment	Dependent	Traffic service can only direct vehicles to street 1 after the water pipe leak is resolved by emergency service.

2.2.2 Spatial and Temporal Range. Most effects of actions are not limited to a single location at a single time. Instead, the effects have *spatial and temporal* ranges of influence, which need to be considered when detecting conflicts. Following the above example, since trucks have to drive on other roads when passing through this area, the trucks will have an effect on the traffic on these roads and the added traffic may cause congestion (i.e., a violation of a performance requirement) for several hours. Also, an increased volume of trucks on a particular day may result in increased release of air pollutants causing air pollution (i.e., a violation of a safety requirement).

2.3 Spectrum of Conflicts

The potential conflicts of smart cities are categorized as device and environmental conflicts as exemplified in Table 1.

2.3.1 Device Conflicts. When more than one action is taken on the same device simultaneously, if these actions are *inconsistent* with each other, they have a *device conflict*. In particular,

- if these actions have opposite directions, but the same numeric parameters, it is an *opposite device conflict*;
- if these actions have different numeric parameters and these parameters cannot be satisfied at the same time, it is a *numeric device conflict*;
- if these actions have the same direction and parameters, but have different durations of application that cannot be satisfied at the same time, it is a *duration device conflict*.

2.3.2 Environmental Conflict. Besides the direct conflicts on shared devices, services are also prone to indirect conflicts caused by unsafe or contrasting effects on the environment (resulting from one or more actions). This is defined as an *environmental conflict*. Environmental conflicts can be categorized into four classes as follows:

- When the set of effects on the environment of a single action causes the state of the city to exceed a safety threshold or violate one/more safety requirements, it results in a *single environmental conflict*.

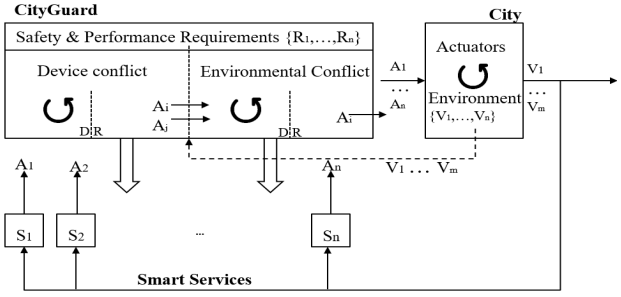

Figure 2: CityGuard Running in a Smart City

- When the additive effects on the environment from two or more safe actions exceed the safety thresholds or violate one/more safety requirements, it results in an *additive environmental conflict.*
- When the effects of multiple actions are opposite on the environment and are not approved to happen by the safety or performance requirements, it results in an *opposite environmental conflict.*
- When the effect of one action is the prerequisite of another action, i.e., multiple actions need to be performed sequentially or concurrently, but the previous one is not taken, it results in a *dependent environmental conflict.*

3 SYSTEM OVERVIEW: CITYGUARD

CityGuard is a safety watchdog built between the smart services and the infrastructure layers (see Figure 1). CityGuard executes as a feedback loop as shown in Figure 2. Variables V_1 through V_m monitor city states, and services S_1 through S_n use the monitored data to choose actions. CityGuard intercepts the actions, decides if there is a device, environment, or both conflicts based on the safety and performance requirements. Unsafe actions and conflicts are detected and resolved through CityGuard. Safe actions are taken in the city and cause the change of city states, which triggers actions from smart services again. As a result, the goal is that only safe actions are executed in the city. However, CityGuard does not guarantee safety, but rather significantly improve it as shown in the evaluation.

The internal structure of CityGuard is shown in Figure 3. There are 4 important components in CityGuard, *City Safety and Performance Requirements (CSPR), City State and Service Action (CSSA), Pre-processing, and Conflict Detection and Resolution (CDR).* Algorithm 1 shows how the components are executed.

3.1 Safety and Performance Requirements Component

The safety and performance requirements component provides the rules for CityGuard to monitor all the actions. It has three modules: (i) principles, (ii) requirements, and (iii) updating requirements. All principles and requirements are defined and specified by city personnel. CityGuard integrates them into the safety checking components.

To start with, CityGuard follows a set of principles to maintain safety requirements. For example, it might contain,

- Any action of a service should not violate predefined city / individual service safety requirements.
- A safety requirement is a function of location and time with conditions.
- When there is a conflict between different safety requirements, follow the city objectives.

CityGuard works with the safety and performance requirements specified by a particular smart city that is running CityGuard. For example, important safety and performance requirements for a smart city might include (i) *Noise levels should be below the following thresholds, community/school (Day 50 db, night 45 db), Mall/working zones (Day 60 db, night 50 db), Highway (Day 70 db, night 55 db)*; (ii) *Actions taken by transportation services should not cause collision of vehicles* ; and (iii) *emergency vehicles should not wait for more than 10seconds at any intersection.*

These requirements, defined in English by the city, are integrated and translated to formal metrics in CityGuard manually. For example, the above requirements can be translated to (i) R_1: $Noise(Location, Time) < xdb$, (ii) R_2: $Num(collision) < 0$ and (iii) R_3: $waitingTime(E) < 10s$.

Since the mapping between actions and safety requirements is not always straightforward, CityGuard simulates and analyzes the effects of an action on the metrics and therefore decides if it is a safe action by examining if the metrics are within their requirements.

Furthermore, with new services added and situations changed in the city, safety and performance requirements are also added and updated.

3.2 Real City State and Service Action Component

City state and service action is considered as the interface of CityGuard to services and cities. It obtains real city states and passes them to CityGuard SUMO in the CDR, obtaining a consistent view of states in the real city and the simulated city. Meanwhile, it also stores and provides the in-coming actions from services and on-going actions in cities.

3.3 Pre-Processing Component

When an action A_i is intercepted by CityGuard, it contains information to direct an actuator, which is also the source for CityGuard to analyze potential conflicts. Usually, an action has the following information, *Device Number* indicates a unique numerical identifier of the actuator on which the action is supposed to be taken. *Service Number* indicates a unique numerical identifier of the service that issues the action. *Act* is the expected action or effect, which depends on the functions of the services and could be a change in states, locations, or send a warning or message, etc. *Duration* is the requested duration of this action. Some actions are continuous actions and need to be acted on for a certain time, while some actions are just one time actions. *Pre-conditions* indicate the pre-conditions of the action, mainly pointing to the essential concurrent or sequential actions. The format is <ActionID, Con/Pre>.

In a pre-processing phase, CityGuard checks the above action information and deals with any missing information. Device number

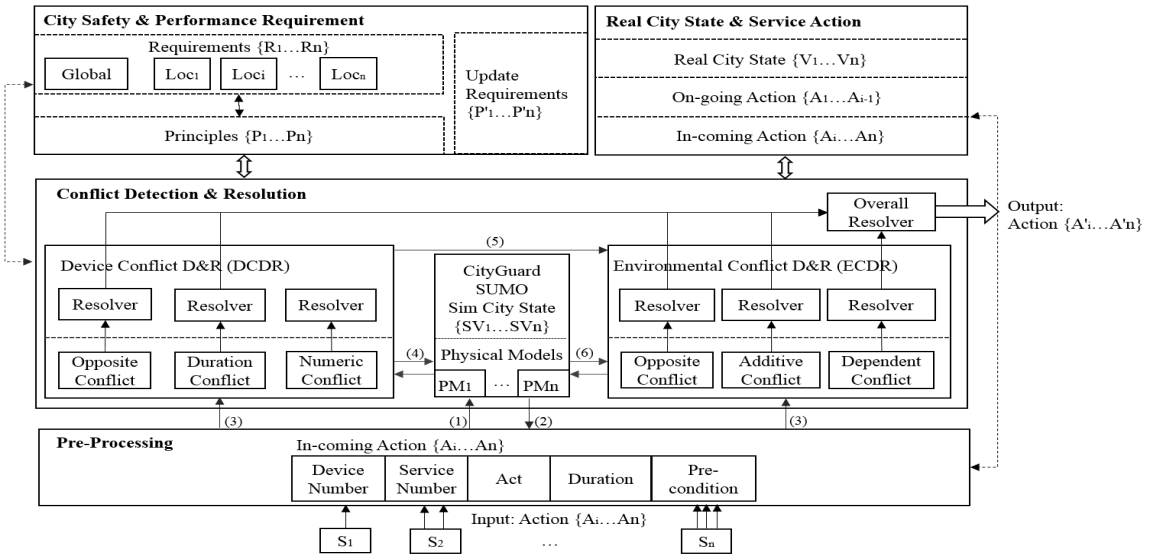

Figure 3: CityGuard Structure (*Pre-Processing* intercepts and checks the safety of the single action, *Conflict Detection & Resolution* detects conflicts among actions after Pre-processing, and *City Safety & Performance* and *Real City & Service Action* store the safety requirements and city states, respectively. Section III describes each component in detail.)

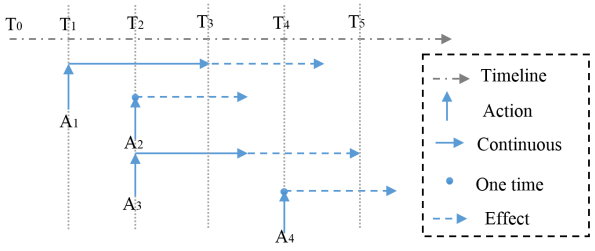

Figure 4: Effects of Actions Taken in the Smart City Over Time

and service number are easy to tell from the source and destination of the action. All *acts* are assumed to be contained in the action information, otherwise, it cannot be executed by the actuator. Unless specified, the duration is treated as 0 if it is missing on the action information. It is a reasonable way to deal with missing duration because even if a continuous action is being treated as a one-time action, it still has an effect on the city performance, which will be detected by the environmental conflict detection procedure.

After intercepting the actions, CityGuard also needs to consider the timing of the actions. CityGuard defines three types of actions based on their action and effect times, including In-coming actions, On-going actions, and Past actions. An In-coming action is one just being intercepted and going to be checked by CityGuard. An on-going action is the one that has been already checked by CityGuard and is running. Because it is still using the device, the in-coming action which wants to take different action on that device may be conflicting with it. A past action is the one that has been taken and finished. Though it may still have an effect on the city, its effects are reflected by the city environmental states. Therefore, CityGuard does not track these actions any more. For example, in Figure 4, at T_2, A_1 is an on-going action, A_2 and A_3 are in-coming actions. Thereby, all of A_1, A_2 and A_3 may be conflicting and need to be checked by CityGuard. However, at T_4, when A_4 comes in, there is no on-going action, then CityGuard only needs to check if T_4 is a safe action in terms of the single action environmental conflict.

As a result, three key parameters are retained by CityGuard, i.e., In-coming actions $\{A_i, ..., A_n\}$, On-going actions, and the City States $\{V_1, ..., V_{i-1}\}$.

Three steps are performed in the pre-processing,

Step 1: Intercept actions and obtain their key information.

Step 2: Check single actions' safety by running them in CityGuard SUMO. Send back unsafe actions with warnings to their services. ((1)(2) in Figure 3)

Step 3: Check Device Number of in-coming and on-going actions, then send the actions with same device number to DCDR because they have a potential device conflict. Pass the other actions to the ECDR. ((3) in Figure 3)

After pre-processing, all the actions sent to the conflict detection components are safe single actions.

3.4 Conflict Detection and Resolution Component

Conflict Detection and Resolution Component consists of 4 sub components, i.e., CityGuard SUMO, Device Conflict Detection and Resolution (DCDR), Environmental Conflict Detection and Resolution (ECDR), and an Overall Resolver (OR).

CityGuard SUMO is the central component of the CDR which is used by both DCDR and ECDR to simulate the effects of actions on a real city. The solution uses the Simulation of Urban MObility (SUMO) [7], a traffic simulation that models inter-modal traffic systems including road vehicles, public transport, and pedestrians.

Algorithm 1: CityGuard

input : Action Set $\{A_k\}$
output : Safe Action Set $\{A'_k\}$
initialize : CityState $\{V_k\}$, SimulationState $\{SV_k\}$, Requirement $\{R_k\}$, SimuStep = 0
while $\{A_k\}$!= 0 **do**
  Pre-Processing:
  for *action = 1: length*($\{A_k\}$) **do**
    $\{SV_k\} = \{V_k\}$;
    for *SimuStep = 1 : n* **do**
      $\{SV'_k\}$ = CityGuardSUMO (A_k, $\{SV_k\}$);
      f_{FC} = SafeCheck $\{SV'_k\}$, $\{R_k\}$;
      if $f_{FC} == 1$ **then**
        A'_k = Resolve(A_k);
      end
    end
  end
  DevcieConflict:
  f_{DC} = DeviceCheck ($\{A'_k\}$);
  if $(f_{DC_{ij}} == 0) \| (f_{DC_{ij}} == 1 \&\& A_i == A_j \&\& Dur(A_i) = Dur(A_j))$ **then**
    go to EnvironmentConflict
  end
  if $f_{DC_{ij}} == 1$ **then**
    if $A_i == -A_j$ **then**
      A'_k = Resolve(A_i, A_j)
    end
    if $A_i + A_j > R_k$ **then**
      A'_k = Resolve(A_i, A_j)
    end
    if $Dur(A_i)! = Dur(A_i)$ **then**
      A'_k = Resolve(A_i, A_j)
    end
  end
  EnvironmentConflict:
  $\{SV_k\} = \{V_k\}$;
  for *SimuStep = 1 : n* **do**
    $\{SV'_k\}$ = CityGuardSUMO (A'_k, $\{SV_k\}$);
    f_{EC} = check ($\{SV'_k\}$, $\{R_k\}$);
    if $f_{EC} == 1$ **then**
      A'_k = Resolve(A_i);
    end
    if $Ef(A_i) == -Ef(A_j)$ **then**
      A'_k = Resolve(A_i, A_j)
    end
    if $A_i \leftarrow A_j$ *&& exist*(A_j) *== 0* **then**
      A'_k = Resolve(A_i)
    end
  end
  $\{A'_k\}$ = OverallResolver$\{A'_k\}$;
end

By implementing smart services and simulating real city scenarios in SUMO, CityGuard SUMO plays an important role to test the primary and secondary effects of actions. To do this, there are different Physical Models (PM) in CityGuard SUMO to simulate the primary and secondary effects of actions. For example, the traffic - air PM knows how the numbers and speeds of different types of vehicles affect emissions (e.g., CO, HC, PM) quantitatively. As a result, the secondary effects on environments of actions from transportation services are obtained.

Before executing, CityGuard inputs into the simulator the same states of the city where actions are going to be taken. Then, it runs one or multiple actions in this scenario into a future time interval. New states of the city after taking these actions are sent back to DCDR and ECDR, where the decisions of detection and resolution conflicts are made. SUMO also has models to simulate accidents.

3.4.1 Device Conflict Detection and Resolution Component. This component is shown in the left part of CDR in Figure 3. Three types of device conflicts are processed using different detectors and resolvers with corresponding models. Once potential device conflict actions are received by DCDR, their *acts* are sequentially checked by opposite, duration, and numeric conflict detection modules with the steps described below.

Step 1: Following the logic defined in Section II, DCDR compares the given actions to detect if they are (i) opposite, (ii) with different numeric requests, or (iii) the same. Accordingly, proceed to Steps 2 through Step 4.

Step 2: If they are (i) opposite, call the opposite conflict resolver, which makes a decision according to ORR.

Step 3: If they are (ii) with different numeric requests, whether they are conflicting depends on if they can be taken at the same time, which is simulated in CityGuard SUMO. If one action with the larger numerical request actually tolerates the others, there is no numeric conflict because all of the actions can be satisfied. Otherwise, the numeric resolver is called for decision making according to ORR.

Step 4: If they are (iii) the same, compare their *durations*. If durations are the same, there is no device conflict between/among them and the actions are sent to ECDR; If durations are different, similar to Step 3, actions are simulated, and tolerance is checked. A longer duration is accepted by its resolver if they are tolerant. Otherwise, a decision is made according to ORR.

Step 5: All actions with a marked decision from DCDR are sent to ECDR.

3.4.2 Environmental Conflict Detection and Resolution Component. All the actions received by this component (shown on the right side of CDR in Figure 3) from the Pre-processing and DCDR components are sent to run in CityGuard SUMO for N steps to see their combined effects on the environment (see (6) in Figure 3). Through analyzing performance results from CityGuard SUMO and checking with the safety and performance requirements, ECDR detects three types of environmental conflicts with following steps.

Step 1: Check if their additive effects conflict with the city requirements, which includes the situations when simulated states exceed required thresholds and when forbidden unsafe cases happen. If so, then there is an additive conflict. The additive resolver is called.

Step 2: With the results from CityGuard SUMO, the environmental opposite conflicts are detected by comparing the primary effects detected from pre-processing component with the combined effects from ECDR. If there is an opposite conflict detected, actions are sent to the corresponding resolvers. If these opposite effects on the environment violates safety requirements, a resolver resolves it according to ORR. Otherwise, the resolver decides whether to resolve it or not based on the context of the system.

Step 3: In the dependent conflict detection model, pre-conditions of actions are checked.

Step 4: All marked actions are sent to the overall resolver.

After DCDR and ECDR, there is a set of actions with marked temporary decisions. If a single action violates the safety requirements, it is rejected in the pre-processing stage. However, actions which are detected to have a device conflict or an environmental conflict receive a temporary decision from each part, respectively. In these cases the final decision is made from the Overall Resolver. It is necessary for two reasons, (i) generally, DCDR and ECDR proceed in parallel and don't intervene with each others' decisions, and (ii)

it is able to avoid rejecting both actions accidentally. For example, actions rejected by one component because of conflicting with another action, which may also end up with a rejection in another resolver in next step. With an overall resolver, a final decision is made by considering the decisions from all the resolvers.

In the *Overall Resolver*, there is a set of actions with temporary decisions marked for final decision making. Actions approved by both DCDR and ECDR are considered as safe actions and can proceed in the real city. Actions with both device and environment conflict rejections are rejected along with a rejection message sent back to their services. Furthermore, actions marked rejection by ECDR are considered as unsafe actions to be rejected, while actions marked with a device conflict rejection are re-checked to see if they still have conflicts with other approved actions. If not, it is viewed as a safe action as well. Otherwise, it is rejected.

In addition, resolvers in the DCDR and ECDR sub-components follow the *Overall Resolution Rules (ORR)* for decision making. There are two principles, priority-based and performance-based, to make decisions when it concerns conflicting actions from multiple services.

First, different domains have a priority based on their importance to the city, giving services from each domain the same priority. An example priority is Safety > Emergency > Environment > Traffic. However, different cities may value different domains.

Second, actions of the services from the same domain are decided by their performances in CityGuard SUMO. In the pre-processing component, each action is tested in the simulation and comes back with a performance result. If two actions in the same domain conflict with each other and they are not unsafe, the one with better performance is accepted while the other one(s) is rejected.

4 SIMULATION AND EVALUATION

CityGurad is evaluated using a smart city simulator, which is extended from SUMO, a transportation simulator. The evaluation uses a real map of one-half of Manhattan, New York City and generates simulated scenarios based on real data. Ten smart services from the domains of transportation, emergency, and environment are implemented and installed in 20 major locations. These services are listed in Table 2. Due to the limitations of SUMO, only services from the above domains are implemented. However, these are examples of the most common smart services running in real cities and are both representative and important.

4.1 Initialization and Metrics of Simulation

In order to simulate the performance of CityGuard in a real city environment, to start with, real city data from Manhattan is analyzed. From Traffic volume counts of New York city data [4], the traffic volumes from 160 streets in Manhattan during 2013-2014 are obtained. It is calculated from the data set that the average traffic volume for all streets is 105,397 vehicles and 658 vehicles per street per hour, making the in-coming vehicle rate as 5.5s per vehicle.

In order to generate a scenario closest to the real traffic pattern in Manhattan, three steps are performed to configure the simulation. First, we selected the average traffic volume data of Manhattan from 8:00 am to 2:00 am to generate the traffic data stream in the simulation. Second, the traffic streams for main streets of New York

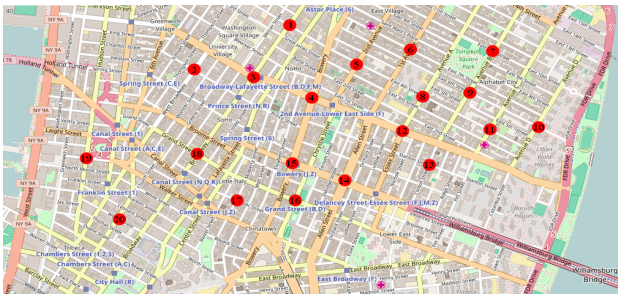

Figure 5: Simulated Manhattan with 10 services running at 20 different locations (denoted as red points).

city, such as the Bowery, Allen Street, and Broadway, are set based on their own average traffic volume per street. Finally, stream data for other streets follow the average volume for the entire Manhattan area, i.e. the in-coming rate of 5.5 s/vehicle is used. In this way, the lower half of Manhattan including 102 streets and 454 traffic lights are used as the **platform** for all simulations in the evaluation.

Furthermore, important safety and performance **metrics** are chosen for evaluation, as shown in Table 3. The first four metrics are obtained from internal models of SUMO, indicating the transportation safety and performance. For instance, the possibility of a collision happening increases when the density of traffic increases and the distance between vehicles shrinks. Meanwhile, these metrics when applied to emergency vehicles also indicate the performance of the emergency domain. Moreover, mean speed, waiting number and waiting time per lane are measured for transportation performance. Noise, CO, HC and PM_x are measured for environmental performance and are considered as safety metrics.

For the evaluation, the requirements shown in Table 4 are assumed to be specified for the smart city.

To better understand the complexity and scope of conflicts, the pre-processing component is first evaluated in isolation. This demonstrates the spatial and temporal effects of individual service. Next, the overall CityGuard is evaluated.

4.2 Overall Evaluation

4.2.1 Pre-processing. In the Pre-processing component, single actions are intercepted and their primary and secondary effects on the environment are simulated to see if there is any violation of safety or performance requirements.

Spatial and temporal ranges of action effects are tested by CityGuard SUMO on 10 services in 20 locations. At first, the city is simulated without services and the metrics for all the streets near the services are recorded as the baseline. Each service is simulated individually in different locations, and the same states are recorded and compared with the baseline. If the variance is above 5%, it is viewed as an effect on the streets from this action. In this way, the number of blocks away from the service block that are affected by the action is identified for each service in all locations. Similarly, how long the effects last on the environment are also monitored.

The results are shown in Table 5, the first column are the metrics, the second column lists services running with and without CityGuard, the third column is when there are no services at all,

Table 2: Services running in Simulated Manhattan

No.	Service	Domain	Description
S1	Congestion Service	Transportation	Its purpose is to minimize traffic congestion. When the waiting number of vehicles on the lane exceeds 50% of its total tolerance, it will adjust the traffic signal to release congestion.
S2	Pedestrian Service	Transportation	Its purpose is to minimize waiting time of pedestrians. When more than 2 pedestrians press crossing button, it will shorten their waiting time by adjusting traffic signals.
S3	Vehicle Navigator	Transportation	Its purpose is to release vehicles from traffic congestion. When there is a traffic congestion or closed lane causing a vehicle waiting for a long time, it will call the re-route function to choose the next shortest path to its destination.
S4	Air Pollution Control	Environment	Its purpose is to control air quality level with emphasis on the CO and HC gas released by vehicles on streets. It will limit the number and speed of vehicles when air pollution level is high by adjusting traffic signal and sending speed request to vehicles directly.
S5	PM2.5/PM10 Control	Environment	Similar to S4 with emphasis on the PM2.5 and PM10 in the air.
S6	Waste Management	Environment	Its purpose is to manage waste in cities by sending out waste pickup vehicles regularly.
S7	Noise Control	Environment	Its purpose is to control noise pollution causing by traffic. When noise level exceeds its threshold, it will control the number of vehicles going through related streets and redirect vehicles on the streets by adjusting traffic signals.
S8	Event Service	Environment	Its purpose is to ensure smooth operation of a city event by blocking the lanes nearby the event.
S9	Accident Service	Emergency	Its purpose is to take the first action to block the adjacent areas of a traffic accident.
S10	Emergency Service	Emergency	Its purpose is to minimize the waiting time of emergency vehicles. When there is an emergency vehicle waiting in the lane, it will adjust the traffic signal to let it go through immediately.

Table 3: Metrics for Evaluation of City Performance: S=Safety and P=Performance Metrics

Name	Description
Jam (P)	Number of cases when a vehicle can not continue because there was no space on the next lane
Yield (P)	Number of cases when a vehicle is unable to cross an intersection where it did not have priority
Collision (S)	Number of cases when a vehicle violated its minimal distance requirement in relation to its leader vehicle
Wrong Lane(S)	Number of cases when a vehicle was unable to move because it could not continue its route on the current lane and was unable to change to the correct lane
Mean Speed (P)	The mean speed of the vehicles on the specific lane (km/h)
Waiting Number (P)	The number of vehicles waiting on the lane, a speed of less than 0.1 m/s is considered a wait.
Waiting Time (P)	The time that a vehicle waits on the lane
Noise (S)	The noise emitted by the vehicles on the specific lane (dB)
CO (S)	The complete amount of CO emitted by the vehicles on this lane during the actual simulation step (mg)
HC (S)	The complete amount of HC emitted by the vehicles on this lane during the actual simulation step (mg)
PM_x (S)	The complete amount of PM_x emitted by the vehicles on this lane during the actual simulation step (mg)

Table 4: City Safety Requirements

Transportation:

- Actions should not cause collisions of vehicles.
- Vehicles should not be directed to travel in the wrong direction or to blocked roads.
- Traffic signal lights should follow safety logic.
- Vehicles from orthogonal directions should not cross an intersection simultaneously.
- Actions should not increase traffic congestion by more than 10% .
- Actions should not increase Yield by more than 15% .
- Actions should not increase waiting time of emergency vehicles by more than 10%
- The number of waiting vehicles in a lane should be less than the maximum vehicle capacity of the lane.

Emergency:

- Emergency vehicles should not wait for more than 10s at a intersection.
- Emergency vehicles should not be directed to a blocked lane or area.

Environment:

- Action should not create more than 50 dB noise per lane.
- Action should not emit more than 50 mg CO per lane.
- Action should not emit more than 1 mg HC per lane.
- Action should not emit more than 0.2 mg PMx per lane.

and S1 to S10 are the data from just one service running. Following insights are obtained from the results that are presented Table 5.

When one service improves one aspect of city, its secondary effects may have a negative influence on other metrics. If not controlled, this influence may exceed the safety and performance requirements.

When there is no CityGuard, actions from 5 services cause collisions, which affect city safety significantly and even create more serious secondary effects. These collisions are prevented by CityGuard. Meanwhile, the number of jam and yield violations also exceeds the threshold of safety requirements, which are highlighted in Table 5. However, with control of CityGuard, jam and yield from all actions are controlled under the safety requirement.

Another key metric for safety performance, waiting time of emergency vehicles (Row 8), is increased significantly by 8 services , exceeding the threshold of safety requirements. With CityGuard it is controlled to under 10s. For example, waiting time of emergency vehicle is reduced from 19.3s to 9.3s by CityGuard, improving the performance by 101%.

It is important to notice, that in these 8 cases, comparing the performance of other metrics when CityGuard runs with no services in the city, it improves the city's performance, i.e., services' functions are not affected by CityGuard. For example, air pollution service (S4) increases the waiting time of emergency vehicles to 15s without CityGuard and to 10s with CityGuard, and where, CO are 7.8mg and 7.6mg, respectively. Comparing with 11.74mg of CO release without any service, S4 with CityGuard improves the air quality. As a result, with CityGuard, air quality is improved without affecting emergency vehicles. In some cases, service performance on one or two metrics is compromised with CityGuard because some of its unsafe actions are rejected. However, CityGuard's role here is to obtain a safe environment while helping services improve the city. For example, in column S1, waiting time for normal vehicles is 98.5 seconds without CityGuard, while 100.2 with CityGuard. Though this performance is compromised by 2 seconds, it still improve the transportation performance comparing with the one without service (121.82 seconds). Most importantly, the waiting time for emergency vehicles decreases from 11.5 to 10 comparing the results without and with CityGuard, consequently S1 is controlled to work under safety requirements.

If a service does not violate requirements at all, it will not be affected by CityGuard, such as S3 and S6.

Some additional observations from these simulations are:

- The ranges of spatial and temporal effects vary by functions of services and locations. For example, Event Service (S8) usually has a longer effect time than the Congestion Service (S1). Effect range at Location 15 is always larger than that at Location 20.

Table 5: Effects on City with single services running

Metrics		No S	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Jam	No CG	102	100	**422**	60	102	68	102	84	**402**	**222**	**290**
	CityGuard	-	100	**110**	60	102	68	102	84	**86**	**120**	**120**
Yield	No CG	192	230	**400**	178	226	190	194	210	**622**	**530**	**598**
	CityGuard	-	218	**212**	178	200	190	194	198	**202**	**196**	**218**
Collision	No CG	0	0	**4**	0	**2**	0	0	0	**4**	**4**	**6**
	CityGuard	-	0	**0**	0	**0**	0	0	0	**0**	**0**	**0**
Wrong Lane	No CG	22	22	40	6	14	12	22	20	**82**	42	**46**
	CityGuard	-	20	20	6	14	12	22	20	**20**	20	**20**
Mean Speed	No CG	11.28	12.36	10.29	13.64	9.8	10.1	11.28	12.21	8.98	9.14	9.11
	CityGuard	-	13.22	11.1	13.64	10.6	11	11.28	13.5	10.89	10.7	8.99
Waiting Number	No CG	1.66	0.71	**5**	0.72	2.1	2.21	1.67	1.7	**4.71**	**3.98**	1.89
	CityGuard	-	0.73	**1.7**	0.72	1.8	1.91	1.67	1.65	**1.69**	**1.8**	1.86
Waiting Time	No CG	121.82	98.60	**311.7**	101.5	**150.1**	**149.4**	121.9	136.1	141.3	157.6	181.4
	CityGuard	-	100.2	**130.97**	101.5	**129.41**	**134.87**	121.9	123.1	131.8	137.2	134.7
Waiting Time (E)	No CG	9.5	**11.50**	**13.3**	**10.93**	**15**	**13.5**	9.5	**15.3**	**17.1**	**19.3**	3.2
	CityGuard	-	**10**	**9.4**	**10**	**10**	**9.6**	9.5	**10**	**9.8**	**9.6**	3.41
Noise	No CG	33.34	34.98	40.12	32.1	31.5	32.5	33.34	17.6	37.6	39.6	39.4
	CityGuard	-	32.14	35.6	32.1	30.93	31.1	33.34	18.7	35.6	35.1	39.6
CO	No CG	11.74	10.85	**16.9**	10.65	7.8	7.41	11.74	11.61	13.4	14.11	13.8
	CityGuard	-	10.75	**12.4**	10.65	7.6	7.31	11.74	10.71	12.31	13.13	12.3
HC	No CG	0.49	0.46	0.71	0.48	0.23	0.29	0.49	0.51	**0.69**	**0.91**	0.65
	CityGuard	-	0.46	0.53	0.48	0.22	0.27	0.49	0.46	**0.64**	**0.69**	0.56
PMx	No CG	0.08	0.07	0.12	0.69	0.05	0.03	0.08	0.08	0.11	**0.17**	0.11
	CityGuard	-	0.07	0.09	0.69	0.05	0.03	0.08	0.08	0.1	**0.13**	0.09

(1)Without CityGuard (2)With CityGuard

Figure 6: Performances of traffic at the intersection between Bowery and Kenmare with and without CityGuard. Yellow and red objects denote regular and emergency vehicles. With CityGuard, there are higher traffic flow, less congestion, and emergency vehicles are prioritized for faster travel time.

- The effects are more significant on the streets of the intersection where services run, rather than the neighboring streets.
- The Accident Service (S9) has the largest average spatial effect range of up to 5.6 blocks while the Noise Service (S7) has the lowest one of 0.8 blocks. The average effect range for all 10 services is 2.3 blocks.
- Temporal effects of different services vary significantly not only by service type and locations, but also by the specific contexts of the events. For example, the effect range of services is larger in area and longer in time when the traffic density is heavier.

4.2.2 Device Conflict. After Pre-procesing, actions on the same device (S1, S2, S4, S5, S7 and S10) are sent to the component of DCDR. Device Conflicts detected between each two of them in 1 hour are shown in Table 6, which indicates how many times Service A (services in the column) is triggered and how many times it is conflicting with Service B (services in the row). For example, the third item in the second row, "20313/42048" indicates that out of the 42048 times Service S1 is triggered, 20313 times it has a conflict with service S2.

It is demonstrated that (i) conflicts occur very frequently in the city, e.g., 20313 conflicts happened between S1 and S2 and 31312 times between S1 and S4. (ii) Percentages of device conflicts are very high, e.g., device conflicts between S1 and S3 is 74.5% while that between S7 and S2 is 68.6%.

Device conflicts can cause serious consequences in the city if there is no prevention. For instance, in Figure 6, without CityGuard, actions from emergency service and traffic congestion service are conflicting, causing serious traffic congestion. More importantly, an emergency vehicle is trapped by this congestion. This violates safety requirements.

4.2.3 Environmental Conflict. In the environmental conflict component, integrated effects of concurrent actions are tested in simulated Manhattan. The relationship between environmental conflicts detected and the number of services running in the city is analyzed. N services are randomly chosen for 50 times and average performance is recorded. Performance of representative metrics for safety and performance, i.e. number of collisions, number of jams, CO air pollution, and traffic waiting time per lane are shown in Figure 7, leading to the following observations.

- Generally, with the growing number of services running in the city, effects on the environment become worse if there is no safety protection mechanism.
- The number of collisions increase significantly when more than 7 services are running together, which seriously violates city safety. However, there is no collision with CityGuard.

Table 6: Pairwise Device Conflicts Detected in 1 hour

	S1	S2	S4	S5	S7	S10
S1	-	20313/42048	**31312/42048**	102/42048	291/42048	3043/42048
S2	20313/54475	-	108/54475	62/54475	653/54475	5321/54475
S4	31312/42588	108/42588	-	79/42588	439/42588	4987/42588
S5	102/745	62/745	79/745	-	110/745	311/745
S7	291/952	**653/952**	439/952	110/952	-	422/952
S10	3043/8217	**5321/8217**	**4987/8217**	311/8217	422/8217	-

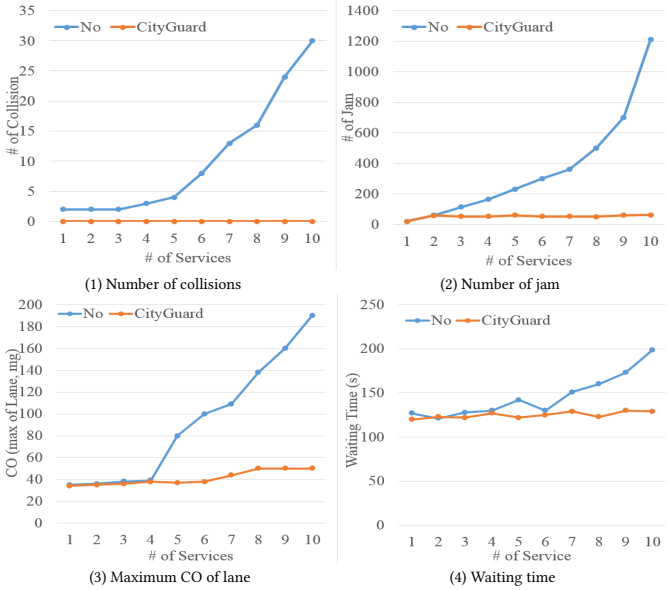

Figure 7: Comparison of Service Effects on Environment with and without CityGuard

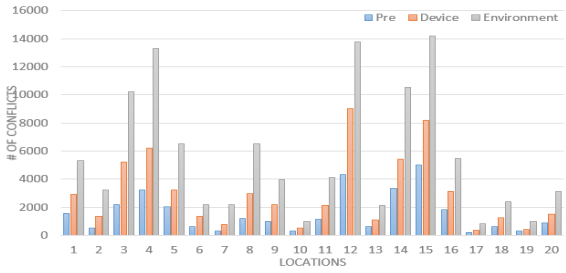

Figure 8: Number of conflicts detected from each location

- Similar to the number of the collisions, the number of Jams is also affected by the number of services running, which can be controlled under a performance threshold with CityGuard running.
- Although waiting time is not affected as drastically as other metrics, it is seen from Figure 7 (4) that CityGuard can help the city to obtain a relatively stable performance with increasing number of services. When the number of services is 10, the waiting time is improved by 35%.
- CityGuard improves Air quality (as measured by emission of CO) over services without CityGuard by 51.3% when 5 services are running, and by 73.7% when 10 services are running.

The total number of conflicts detected are summarized by 20 locations of services in Figure 8. In all locations, environmental conflicts have the highest percentage, which usually is at most twice the percentage of device conflicts and as much as 4 times as single unsafe actions. Moreover, the number of conflicts vary from location to location. The busier and larger an intersection is, the more conflicts occur there.

5 DISCUSSION AND LIMITATIONS

Safety-aware conflict detection and resolution are critical and complicated issues in smart cities, which is difficult to be solved at once. CityGuard is a watchdog solution to dynamically improve the safety of smart cities by focusing on actions that impact the environment and act across services. It does not and we argue fundamentally, cannot apriori guarantee complete safety due to the dynamics and uncertainty in the real world. It is a general architecture which can be integrated into any Smart City IoT platform. Being the first work of its kind, CityGuard has limited scope in the following issues.

Safety Requirements: In this paper, it is assumed that actions and requirements that are integrated with CityGuard have a uniform format. For example, it is assumed that actions provide necessary information, i.e., device, act, duration, and pre-condition(s). The mapping of actions to the safety and performance requirements were performed manually, because the existing safety rules of cities are defined in English by different government departments. There are no tools to translate them into code automatically. However, this manual mapping process doesn't affect the generalizability of CityGuard. Also, if new requirements arise, the module called *City Safety & Performance Requirement* can update the requirements maintained in CityGuard and detect conflicts using the new requirements (See Figure 3). In the future, we will explore ways to specify requirements and algorithmically map actions to requirements. In this case, semantic token extraction techniques for textual interventions [19] can be useful.

Conflict Detection and Safety: Simulating the primary and secondary effects of an action, CityGuard checks the effects with the city-wide safety requirements to check if there is a conflict among actions. Many such conflicts are found and resolved. However, some smart services have their own internal safety requirements not directly visible to CityGuard. CityGuard cannot always detect the safety violation of internal safety requirements of a service unless they impact the environment in a negative manner as specified in the city-wide requirements. Therefore, although currently CityGuard's goal is to significantly improve the safety of a city, safety violations can still occur. Ideally, in the future, as services are added to a city, all the internal safety requirements should be exported

and compared to the city wide requirements. Any conflicts here should also be resolved.

Conflict Resolution: Currently, CityGuard focuses on the methods for conflict detection rather than providing a comprehensive solution for conflict resolution. Only predefined priority based resolvers are used to resolve conflicts once they are detected. However, in real scenarios, conflict resolutions can be much more challenging and might require considering several contextual factors. The list of such factors includes, but is not limited to, the importance of the action (i.e., that causes conflict) and its service, the effects on the environments and human beings, the cost of rejection, the optimal combinations of conflicting actions, dynamically changing objectives, etc. Thus, a comprehensive solution of conflict resolution demands further research.

Simulation: In the current implementation of CityGuard, the selection of applications and the accuracy of conflict detection rely on the accuracy of the SUMO simulation and its embedded models. This is reasonable as (i) SUMO has been widely used to provide simulations on city mobility with sophisticated physical models developed in research and used in practice for over 15 years (as described in Section 6); (ii) SUMO is only an example tool for CityGuard, which can be updated, combined, or replaced with more sophisticated models/simulators once they are available; (iii) The evaluations in this paper are limited yet realistic to illustrate how we detect conflicts and improve a city's safety and performance. The approach for conflict detection in CityGuard is sophisticated enough to be applied in other domains once accurate models of those domains are available.

Human Factors: When CityGuard simulates the potential primary and secondary effects of actions during its assessment phase, it assumes that the people (i.e., citizens) and the devices comply with the recommended actions. Decisions to accept or reject actions are based on these assumptions. However, the city state evolves based on what actually happens in the city, e.g., a person may not follow the advice or an unexpected / unprecedented event may occur. CityGuard includes a feedback loop to monitor and react to this situation.

Privacy and Security: Privacy and security are of great significance to a smart city. As they are out of the scope of this paper, a brief discussion below highlights the key issues.

There are potentially many different privacy policies in the smart services and for the city as a whole. Policies themselves might conflict with each other or be violated due to actions from services. When actions are taken by services, CityGuard can be extended to match those potential actions with city wide privacy policies and, therefore, detect potential policy conflicts, dynamically. In other words, the core concepts found in CityGuard can be applied to privacy. If services export their privacy policies then conceptually a general policy conflict detection between the city and a service can be detected at installation time.

There are two main levels of security issues in the context of this work, which are the security of smart services and the security of CityGuard itself. The security of smart services should be maintained by themselves. However, if a service has been attacked and it is taking erroneous actions, CityGuard helps in avoiding some safety violations because those erroneous actions might conflict with the safety requirements known to CityGuard and therefore be blocked. In addition, there are ways that CityGuard itself can be attacked, such as approving unsafe actions, blocking actions from other services, setting higher priority for the service, etc. Therefore, security mechanisms must be included in CityGuard prior to real deployment.

6 RELATED WORK

6.1 IoT Platforms for Smart Cities

There are several commercially available IoT platforms for smart cities, such as, IBM Watson IoT [3], Azure IoT suite from Microsoft [2], Intel IoT platform [6], and AWS IoT from Amazon [1]. They provide support for setting up IoT infrastructure customized to application requirements. They address different aspects of potential city-level IoT infrastructure, including but not limited to, scalability of sensing and actuator modules, real-time response, cloud support for IoT, real-time stream analytics, raw data storage, data driven dynamic applications, and network and data security. Although such IoT platforms can be utilized for developing scalable smart city services, they consider smart city applications as independent entities. Hence, they don't focus on the integration of services and its subsequent complexity (e.g., concurrency and conflicts). While the existing IoT platforms provide support for safeguarding connected devices, networks, data transmissions and data accessibility of the IoT infrastructure, none of them addresses the safety challenges introduced by integration of systems/services (e.g., conflicting operations, policy violations, and conflicting effects of services). To the best of our knowledge, we are the first to formulate and develop a **safety-aware architecture** for detecting and resolving potential conflicts in the context of smart cities.

6.2 Safety in Automation Systems

Safety issues have been well studied in Automation systems. Standards and rules for functional safety of automation systems have been made, such as IEC 61508 and ISO 26262 [10], which define functional safety for automotive equipment applicable throughout the lifecycle of all automotive electronic and electrical safety-related systems. They support the product development from hardware, software and system levels, and provide safety analysis. These standards can be very helpful when extended to the development of single smart services in smart cities. However, there are no rules for interactions or conflicts among services/systems.

There is some research on the functional safety among multi-agents in automation systems, such as building automation and control systems. The authors in [16] focus on the functions affecting people's safety, security and health while maintaining the functional safety and system security of both the network nodes and the communication protocols. Resendes et al. present a survey on the conflict detection and resolution in home and building automation systems [20]. Pallottino et al. propose a cooperative policy for conflict resolution in multi-vehicle systems, which rests on the assumption that all agents are cooperating by implementing the same traffic rules [18]. These integrated systems are only within the domains of transportation, homes, and buildings. However, functional safety in smart cities is much more complicated as it involve multiple domains, different types of services and drastically large numbers of actuators.

6.3 Conflict Detection and Resolution

There are some existing systems to detect dependencies across multiple human centric CPS systems. Munir et al. focus on detecting dependencies across interventions generated by different human-in-the-loop apps (e.g., health apps) [14]. They use simulated apps and structured metadata from each app. They rely on a physiological simulator [9] to approximate potential effects of an intervention. Preum et al. developed Preclude, a system to detect conflicts in textual health advice generated from smart phone health applications and health websites [19]. While they provide a taxonomy of conflicts in health advice and solution to detect different types of conflicts, their approach is focused on textual interventions only.

Another relevant system is DepSys that aims at detecting dependencies from multiple smart home apps [15]. It is a utility sensing and actuation infrastructure specifically designed for smart homes that detects, and resolves conflicts among multiple smart home apps by addressing multiple dependencies. SIFT [11], a safety centric programming platform, detects whether apps running in an IoT environment conform to safety policies and whether apps result in logical conflict with each other. Although it detects logical conflicts (equivalent to *opposite* conflict of CityGuard) using rule based approach, it does not consider different nuances of detecting conflicts / policy violations, e.g., effects, secondary effects, emphasis, and conditions, and it is not applied to inter-services problems.

Our previous work proposed a comprehensive typology of conflicts and a watchdog architecture for conflict detection and resolution in the context of smart city applications [12]. It identifies different characteristics of smart city services that contribute to potential conflicts in smart cities, e.g., uncertainty, real-time, dynamic behavior of services, spatio-temporal constraints, duration and scale of effects. It focuses on the runtime device conflict detection and detect conflicts by imposing pseudo-services on historical data.

6.4 City Simulator

SUMO [7] is an open source microscopic traffic flow simulation, which can import net/map and traffic demand modeling components. It has been utilized in several traffic flow related research, such as vehicular communication, route choice and dynamic navigation traffic light algorithms emission and noise modeling person-based Intermodal traffic simulation In our work, SUMO is used as a smart city simulator with help of Traffic Control Interface (TraCI) [21], a technique for interlinking road traffic and network simulators. With TraCI, the behavior of vehicles during simulation runtime can be controlled and thus the influence of actions on the simulated city are better understood.

7 CONCLUSION

CityGuard, a safety-aware novel watchdog architecture for detecting and resolving conflicts in a smart city is developed and evaluated. CityGuard is safety-aware as it focuses on maintaining safety requirements by detecting and resolving conflicts before they occur. In doing so, CityGuard also considers the context of potential conflicts and resolves conflicts to maximize performance metrics. CityGuard is able to simulate the primary and the secondary effects of the actions performed by services, detect and resolve conflicts among those actions, and thus improve safety. Evaluations are performed using a simulation of part of New York City with 10 smart services located in 20 places. Using 6 safety metrics and 5 performance metrics, the evaluation shows that CityGuard is able to minimize safety violations, and improve overall city performance when compared to baseline methods.

ACKNOWLEDGMENT

This work was funded, in part, by NSF under grants CNS-1527563 and CNS-1319302.

REFERENCES

[1] *AWS IoT*. https://aws.amazon.com/iot/.
[2] *Azure IoT Hub documentation*. https://azure.microsoft.com/en-us/documentation/services/iot-hub/.
[3] *IBM Watson IoT Platform*. http://www.ibm.com/internet-of-things/.
[4] *New York City Open Data*. https://nycopendata.socrata.com/.
[5] *Oracle Smart City*. https://www.oracle.com/applications/primavera/solutions/smart-city-projects/index.html.
[6] *The Internet of Things (IoT) Starts with Intel Inside*. http://www.intel.com/content/www/us/en/internet-of-things/overview.html.
[7] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. 2011. SUMO–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind.
[8] Pierfrancesco Bellini, Monica Benigni, Riccardo Billero, Paolo Nesi, and Nadia Rauch. 2014. Km4City ontology building vs data harvesting and cleaning for smart-city services. *Journal of Visual Languages & Computing* 25, 6 (2014), 827–839.
[9] Robert L Hester, Alison J Brown, Leland Husband, Radu Iliescu, Drew Pruett, Richard Summers, and Thomas G Coleman. 2011. HumMod: a modeling environment for the simulation of integrative human physiology. *Frontiers in physiology* 2 (2011).
[10] M Kucharski, A Trujillo, C Dunlop, and B Ahdab. 2012. *ISO 26262 Software Compliance: Achieving Functional Safety in the Automotive Industry*. Technical Report. Technical report.
[11] Chieh-Jan Mike Liang, Börje F Karlsson, Nicholas D Lane, Feng Zhao, Junbei Zhang, Zheyi Pan, Zhao Li, and Yong Yu. 2015. SIFT: building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. ACM, 298–309.
[12] M Ma, S Masud Preum, W Tärneberg, M Ahmed, M Ruiters, and J Stankovic. 2016. Detection of Runtime Conflicts among Services in Smart Cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 1 –10.
[13] Fei Miao, Shuo Han, Shan Lin, John A Stankovic, Desheng Zhang, Sirajum Munir, Hua Huang, Tian He, and George J Pappas. 2016. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering* 13, 2 (2016), 463–478.
[14] Sirajum Munir, Mohsin Y Ahmed, and John A Stankovic. EyePhy: Detecting Dependencies in Cyber-Physical System Apps due to Human-in-the-Loop. (????).
[15] Sirajum Munir, John Stankovic, and others. 2014. DepSys: Dependency aware integration of cyber-physical systems for smart homes. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*. IEEE, 127–138.
[16] Thomas Novak, Albert Treytl, and Peter Palensky. 2007. Common approach to functional safety and system security in building automation and control systems. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*. IEEE, 1141–1148.
[17] Peter Palensky and Dietmar Dietrich. 2011. Demand side management: Demand response, intelligent energy systems, and smart loads. *IEEE transactions on industrial informatics* 7, 3 (2011), 381–388.
[18] Lucia Pallottino, Vincenzo G Scordio, Antonio Bicchi, and Emilio Frazzoli. 2007. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics* 23, 6 (2007), 1170–1183.
[19] Sarah M. Preum, Md Abu Sayeed Mondol, Meiyi Ma, Hongning Wang, and John A. Stankovic. 2017. Preclude: Conflict Detection in Textual Health Advice. In *Pervasive Computing and Communications (PerCom), 2017 IEEE International Conference on*. IEEE.
[20] Sílvia Resendes, Paulo Carreira, and André C Santos. 2014. Conflict detection and resolution in home and building automation systems: a literature review. *Journal of Ambient Intelligence and Humanized Computing* 5, 5 (2014), 699–715.
[21] Axel Wegener, Micha l Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. 2008. TraCI: an interface for coupling road traffic and network simulators. In *Proceedings of the 11th communications and networking simulation symposium*. ACM, 155–163.