From Back-of-the-Envelope to Informed Estimation of Edge Computing Benefits in Minutes Using Castnet

Conference Paper · June 2019

DOI: 10.1109/ICFC.2019.00028

5 authors, including:

Ketan Bhardwaj
Georgia Institute of Technology
15 PUBLICATIONS **104** CITATIONS
SEE PROFILE

Ada Gavrilovska
Georgia Institute of Technology
121 PUBLICATIONS **1,211** CITATIONS
SEE PROFILE

From Back-of-the-envelope to Informed Estimation of Edge Computing Benefits in Minutes Using Castnet

Harshit Daga, Hobin Yoon, Ketan Bhardwaj, Harshit Gupta and Ada Gavrilovska
College of Computing, Georgia Institute of Technology, Atlanta, GA
{harshitdaga, hobinyoon, ketanbj, harshitg}@gatech.edu, ada@cc.gatech.edu

***Abstract*—Multi-access Edge Computing (MEC) is emerging as the next evolution in computing infrastructure. However, broad adoption of edge computing solutions is still lagging behind. We posit that a significant factor contributing to this trend is related to the challenges in evaluating the trade-offs offered by edge computing and answering a simple question: Will the edge investments be offset with commensurate gains in performance metrics such as improved application responsiveness or lower backhaul bandwidth costs?**

Existing simulation tools, developed for cloud computing and networking research, require a large amount of effort to be applied to the edge computing space. Their native abstractions and interfaces pose limitations in how such frameworks can be extended to allow for characterization and analysis of diverse edge computing use cases. Moreover, as we demonstrate, they can be slow, taking days, to provide insights. To address this gap, we develop Castnet – a fast, extensible, and easy-to-use framework for evaluating the benefits of MEC for different edge deployment models, configuration and functions. We demonstrate that Castnet leads to significant gains in the time-to-insight, while achieving the same fidelity in the results as existing tools. We illustrate the utility of Castnet using two scenarios: investigation of the impact of edge storage capacity for edge-based content caches, and the impact of edge location for an edge-based intrusion detection service.

***Index Terms*—Edge Computing, Edge Cloud, Edge Simulators.**

I. Introduction

In the next decade, the tremendous increase in the number of Internet connected devices [1] will lead to unforeseen amounts of data to be traversing the network [2]. Cisco projects that by 2020 there will be 50 billion devices connected to the Internet and it is estimated that more than 500 zettabytes of data will be generated by such devices, as compared to 135 zettabytes in 2014 [2]. Multi-access Edge Computing (MEC) [3], by providing compute and storage capabilities at the edge of the network, is posed to make it possible to handle that deluge. The expectation is that by leveraging computational infrastructure near the data sources, MEC can reduce both, delays in the data processing time, and the demand on the backhaul network. However currently, the lack of existing edge computing deployments, makes it difficult to systematically study and reason about the benefits, tradeoffs and opportunities enabled by it.

One way to address this gap is through use of realistic emulated testbeds such as SmartSantander [4] and FIT IoT-LAB [5] which provide a heterogeneous environment to conduct experiments under real world conditions. However, their utility in evaluating MEC configurations is bound by the characteristics of the physical testbed, limiting the flexibility to investigate MEC environments with arbitrary configurations and scales. Other efforts take a simulator-based approach to construct full scale models in a controlled environment which otherwise would be either very expensive or infeasible. One approach is to use network simulators such as ns-3 [6]. However, this requires a significant effort in modeling the storage and compute aspect of an edge environment. Another approach is to adopt existing simulation frameworks such as CloudSim [7], SimGrid [8] and iCanCloud [9], which are develop for cloud computing systems. However, the differences in the MEC architecture over a traditional cloud architecture make it hard to use the existing cloud simulators to represent mobile edge computing environments. More recently, iFogSim [10], an extension of the CloudSim framework, proposed to address the gap in simulation platforms for edge computing. Although iFogSim provides modelling support for an edge infrastructure, its design and interfaces make it challenging to configure and evaluate diverse edge computing scenarios, resulting in considerable effort and time required to obtain experimental insights.

Thus, there is a need for a fast, extensible, and easy-to-use tool which can be of practical use for modeling and evaluation of the tradeoffs afforded by edge computing-based solutions in different scenarios.

Technically, the shortcomings of existing simulators stem from the level of abstraction they operate on. For instance, iFogSim abstracts compute and network with cycle accurate fidelity at the expense of increased time to produce answers. ns-3 operates at the network level and lacks native abstractions to describe application-level functionality which is to be carried out along the network paths. The level of abstraction mandates their design decisions. For instance, iFogSim has to be designed using a single event queue, owing to its requirements on fidelity to each abstracted component, which makes it inefficient for larger scale simulations. Similarly, ensuring fidelity to network artifacts makes it difficult to implement high level application behavior in ns-3. We posit that for the task at hand, i.e., gauging benefits of deploying edge functions, both abstraction levels are not suitable.

In this paper, we present *Castnet*, an extendable, lightweight, fast and easy to use edge simulation framework. The high level of abstraction lets Castnet supports all of the relevant entities in an edge computing scenarios – different edge infrastructure deployment models, different edge infrastructure characteristics (latency, compute and storage)

978-1-7281-3236-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ICFC.2019.00028

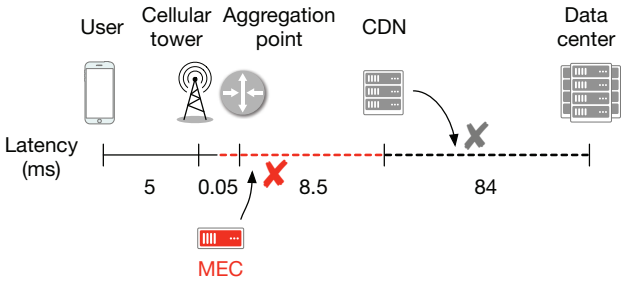


Fig. 1: Mobile network latency breakdown.

and arbitrary logic in the edge applications. Castnet delivers better flexibility through component-based design, i.e., each simulated entity becomes self-contained and has fewer dependencies on others, making the overall analysis performed by Castnet efficient and scalable when compared to existing edge simulators [6], [10].

Further, Castnet's modular design and easy-to-specify interface, make it easy to use and highly customizable, to suit the needs of different user groups. For instance, Castnet can be easily used to estimate tradeoffs and benefits in scenarios where a developer is evaluating a new caching algorithm for their existing edge deployment, an edge infrastructure provider is making capacity planning decisions, or a edge equipment vendor is advertising their products to a new customer.

In summary this paper contributes the following:

1. Castnet – an open-source[1], comprehensive and extensible edge simulation framework for exploring the trade-offs and opportunities in utilizing the emerging MEC with minimal effort. It provides abstractions to extend and model MEC components that could easily be extended to represent concrete edge infrastructure and functions, based on the requirements.
2. Castnet is evaluated with different types of workloads derived from realistic use case scenarios. The evaluations demonstrate that Castnet is lightweight, scalable and is faster than existing solutions. For instance, for a simulation with 15-40 requests/s served per edge node, ns-3 took upto 44% more time compared to Castnet while iFogSim was unable to complete the simulation even after a long time. Castnet also reduces the effort, in terms of lines of code needed to configure an experiment, by 84% when compared to iFog and ns-3.
3. We demonstrate that Castnet can be used for different types of workloads and scales, even with a typical desktop class machine, something that cannot be accomplished with existing tools. This advances the current ability to evaluate the benefits edge computing, which is important for driving its adoption.

II. Background

Figure 1 depicts a generic MEC architecture where the edge nodes can be used to host applications (edge functions [11]) on behalf of existing web services, with the goal of providing performance benefits. These edge functions may incorporate different classes of functionality:

- Acceleration: With a goal of improving the perceived latency performance by splitting or compressing the data sent over the network.
- Aggregation: With a goal of reducing bandwidth usage by filtering the data generated from connected devices by removing the collection, pre-processing and/or filtering of redundant data before sending to the cloud.
- Caching: With a goal of reducing user perceived latency and bandwidth usage by hosting caching servers or streaming services that could benefit web services like web pages requests or video delivery [2].

MEC, by taking advantage of its close proximity to end users has the potential to provide the above services with reduced latency and backhaul data traffic. However, questions regarding the concrete benefits that an edge function can experience when deployed at a specific edge location, and when considering specific workload parameters, remain unanswered. To gain quantitative insights into such questions, we need tool(s) that can be easily extended to characterize the edge function and its deployment scenarios, and that can lead to quick answers with adequate accuracy.

Testbeds are one of the solutions that provide a platform to conduct experiments in a realistic environment. There are numerous such existing testbeds; some examples include SmartSantander [4] and FIT IoT-LAB [5]. The SmartSantander testbed consists of 20,000 sensors deployed in a city, and provides access to the sensor data through a fixed set of APIs supported by their IoT nodes. The FIT IoT-LAB testbed provides bare metal access to all the nodes. It provides access to different smart sensors and hardware capabilities deployed across France. Full access to the nodes helps in reprogramming the nodes and in conducting any desired experiments. [12] provides a detailed survey of available experimental facilities for IoT research.

Although testbeds such as these represent valuable experimental infrastructure, the insights they can provide are limited by the concrete deployment scenarios which can be practically realized on their physical resources. In contrast, simulation frameworks offer flexibility in exploring a broader set of scenarios, and in obtaining insights in relatively shorter amount of time and with less effort in terms of creating and managing experiments.

CloudSim [7] is one of the simulators that is widely used in research. Designed by Buyya et al., the simulator is widely used in data center research on resource management, VM migration, etc. Other such simulators include SimGrid [8], a simulation platform for grid and distributed system, iCanCloud [9], a simulation tool for large scale experiments which can run parallel simulations over a cluster, GreenCloud [13], an extension to a network simulator ns-2, which focuses on obtaining energy consumed by different components of the cloud infrastructure. These tools provide the modeling and simulation support for cloud or distributed server systems. However, they do not natively offer support to model MEC

[1] github.gatech.edu/kernel/castnet

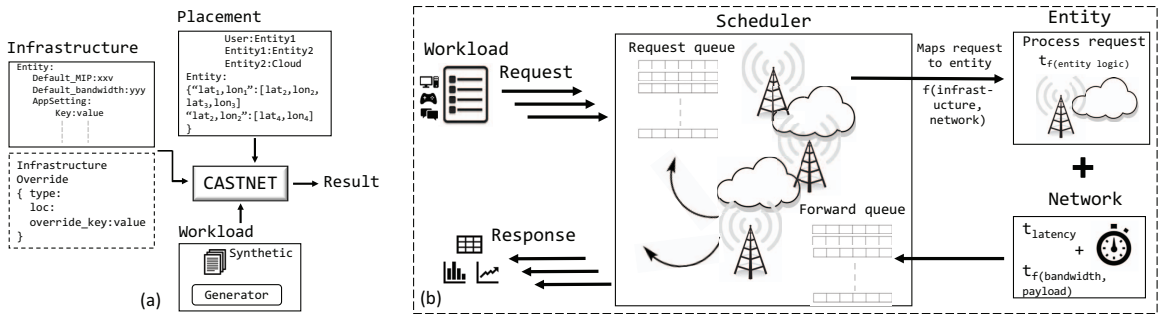


Fig. 2: Castnet overview and internals of a request response cycle inside the simulation.

infrastructure or resources management policies, where an edge can serve as a network, storage or a compute element in the architecture.

iFogSim [10] is an extension of the CloudSim framework built to provide support for experiments designed for MEC. It can be used to investigate the impact of different resource management strategies on application latency or on device energy usage, for configurable topologies of IoT devices, edge node elements and the cloud. iFogSim inherits the internal simulation engine from CloudSim, which is based on a single event queue. The computational requirements of the simulations are tied to the complexity of the queue manipulation operations, and can pose limitations on the types of scenarios which can be practically investigated within a given resource envelop. In addition, configuring experiments with iFogSim requires understanding of the intrinsic details of the framework and significant programming effort.

ns-3 [6], a network simulator, is another framework which can be used to simulate MEC infrastructure. However, unlike iFogSim, ns-3 does not provide native support for representing edge functions beyond native network-level functionality.

Summarizing, the existing simulators are limited in the types of edge configurations, workloads, or applications which they can be used to evaluated. Thus, there is a need for an edge simulator that is fast, simple and easy to use for exploring different MEC scenarios.

III. Castnet

Castnet fills this gap by providing an extensible tool that allows for quick modeling and analyses of the trade-offs of MEC deployment scenarios. Castnet is aimed at edge service and infrastructure providers. Its goal is to provide for rapid estimation of the performance benefits associated with different edge applications, infrastructure configurations, and workload characteristics. Castnet achieves this goal by incorporating a scalable simulation engine, providing for quick insights even as the complexity of the evaluated scenarios increases, and by offering modularized interfaces which reduce the development effort needed to specify different scenario configurations.

Internally, Castnet's simulation engine represents a discrete event simulator where each node in the edge topology is modeled with its own request and forward queue. This design choice is in contrast to the iFogSim design which uses a single shared event queue, to be accessed and manipulated by the events generated by all the nodes in the system. In iFogSim, during a simulation run, various events are recorded across the system and added to the shared queue. The performance of the simulator is directly proportional to the complexity of the queue operations at a given time t, which depend on the size and volume of the generated events. Thus, as shown later in the evaluation section, modeling more complex edge topologies and workload increases the event density, and in turn, the complexity of the underlying queue manipulations, thereby limiting the performance and the scalability of the tool. In contrast, the use of independent queues for each entity in Castnet reduces the queue sizes and the cost of the queue accesses and manipulations, resulting in improved time-to-answer.

Externally, Castnet explicitly exposes a number of abstractions inherent for describing an edge deployment scenario – the edge *infrastructure* topology, the computational elements (*servers*) and the properties of their interconnection links (*network*), the *placement* strategy for the edge function being evaluated, and the *workload*. By explicitly exposing these abstractions, Castnet provides the desired flexibility in specifying and evaluating different MEC scenarios while maintaining simple and intuitive interfaces. In addition, the configurations specified in these modules determine the flow of requests through the edge infrastructure, and guide the simulation engine when evaluating the computational elements and connections along the requests' data paths in order to characterize the end-to-end performance.

Finally, Castnet includes a workload generator used to specify the request pattern seen by edge locations in an evaluated scenario, along with several popular request patterns.

The main components of Castnet are shown in Figure 2 and described in more detail in the remainder of this section.

A. Abstraction Model

Each tier in the MEC architecture can vary in scale, distribution, and positioning. Further, each node within a tier can run discrete applications and can vary in compute, storage and network capacity. To define and simulate the relevant entities, Castnet exposes an *infrastructure model* that uses configuration parameters to determine the location, resource availability and application to be deployed at each node (also called an entity). The edge functions are defined using

Abstraction	Handlers	Description
Infrastructure	`init()`	read and store the infrastructure properties.
	`get()`	provides the entity properties.
Placement	`init()`	creates network links for the topology defined in the config file and creates an entity map with list of server location.
	`get_server_list()`	returns a list containing the server locations for the given entity category.
Network	`set_bandwidth()`	sets the uplink and the downlink bandwidth for the given node.
	`set_latency()`	sets the uplink and the downlink latency for the given node.
	`get_time_taken()`	returns the time taken by given the payload to travel the network.
Server	`create()`	instantiates the resource node with basic configuration information such as server location, allocated compute and storage.
	`init()`	initializing the additional configuration information required by the node. For instance, an edge node implementing caching can include an external configuration such as cache size and caching policy.
	`process_ request/response()`	handlers to server the incoming and outgoing request respectively
	`pre_process_ request/response()`	provides additional hook to perform additional processing before the request or response is been processed.
Workload	`load()`	read or load all the requests from the file.
	`play()`	starts the simulation process and submits the request to the user model if the time stamp of the request is greater or equal to the current simulation time, else it waits.

TABLE I: Describes the important handlers exposed for different abstractions in the Castnet framework.

the APIs provided in the default server class. Next, Castnet uses a *placement model* to abstract the connections between different nodes in the system hierarchy. The placement model specifies the data path of requests (and responses) through the multiple tiers of the infrastructure. The *network model* abstraction provides an easy way to model different bandwidth capacity and the link delays for each network hop based on the experiment requirements and conditions. Finally, *workload model* provides the access pattern to be simulated. Each of these models are provided as an input to the simulator, and are used as explained in III-C.

B. Workload

In order to permit evaluations using request patterns with different properties, Castnet includes a configurable workload generator. Our goal is not to generate an exact request model [14], [15] for an application or set of applications, but to provide a tool that can help in generating a wide range of workload characteristics for MEC following a distribution pattern.

A web service traffic can vary based on the location and time of the day. The request-response sizes and the redundancy in the request content are other variable factors that define the workload. In addition, we studied the request distribution patterns of different web applications which can benefit from edge, and identified two other important dimensions which characterize their request pattern (see Table II). The first concerns the frequency of the requests: dense workloads are request-intensive, whereas sparse workloads exhibit fewer requests arriving at the service for a given time t. The second concerns the time variability of the request pattern: deterministic patterns exhibit regular request distributions, whereas non-deterministic patterns exhibit bursty behaviors. Castnet allows for workloads to be described using all of these parameters.

C. Operation

A Castnet simulation starts by specifying a configuration of the simulation parameters. Castnet provides users with a hierarchical and flexible configuration interface. Figure 2(a) and Listing 1 provide an overview of configurations required by the framework in order to run an experiment, while Table I presents the APIs provided by the framework. The remainder of this section describes the life cycle of a request during its operation.

Category	Example	Characteristic
Sparse Deterministic	Production line [16], Inventory tracking, Environment monitoring [17]	regular, low request traffic due to small scale deployment
Dense Deterministic	IoT camera sensor	regular, high request traffic due to large scale deployment
Sparse Non-deterministic	Parking tracking, Lightning control [18] [19]	irregular, low request traffic
Dense Non-deterministic	Viral video [20]	irregular, high request traffic

TABLE II: Workload categories and their real world use case.

Castnet plays the given workload in a serial manner, i.e., each request is loaded in the order of its timestamp. The request goes through a series of actions for each time interval t until the response is submitted to the client device. The request originates from the client which, through the scheduler, pushes it to the request queue of the connected web edge server entity.

Starting from the lowest tier of the architecture (closest to the client device) and going up to the cloud (data center), the scheduler scans through the request queue of each entity, picks up the elements with the processing time equal to or less than the current timestamp, and submits it to the respective servers. The serial execution helps in distributing the limited resources of a server, such as compute capacity or network bandwidth, among the requests by making it aware of all the requests that are required to be processed at the given time.

The servers process these requests through their application logic, following which the request destination and its new scheduling time is determined. The new scheduling time is generated as a summation of the time taken to process the request by the web server logic, latency between the destination and current server, and the time required to transfer

the payload. The processing time at the server is a function of the compute capacity (i.e., MIPS) required by the request and the compute resources remaining at the server (defined in the infrastructure configuration), while the transfer time taken over the wire is a function of payload and the bandwidth between the entity and the destination location.

If the edge function generates an upstream event (e.g., the same, or a new request is forwarded to the next hop), the event is added to the request queue of the next server (uplink) for further processing. For instance, in case of a cache miss at the edge the request is forwarded to the cloud. A response generated by the edge function (e.g., in case of a cache hit), is added to the forwarding queue of the current server (downlink), and it would eventually be sent in the direction of the client. When the request reaches the client it is logged into a result file. Each log element includes end-to-end break-down of the time spent at each entity in the request path.

D. Limitations

There are limitations on what Castnet can be used for. The edge functions partition the application logic between edge and the cloud [11]. Castnet does not have the capability to automatically perform partitioning, but only provides a mechanism to define edge functions. Further, the accuracy of Castnet relies on the accuracy of the implemented abstractions. For instance, the modules currently used in Castnet do not take into consideration infrastructure contention effects, node failures, or look-up time. Though the accuracy of the results provided by Castnet can be improved in the future through more sophisticated models and extensions of the current design, at present its accuracy is bounded by the accuracy of the underlying component models.

E. Implementation Details

Castnet is written in approximately 3000 lines of C/C++ code, and uses external libraries such as `boost` [21] for performance and `nlohmann json` [22] for parsing JSON.

The synthetic workload is generated using a custom access pattern generator which is implemented in Python. It makes extensive use of NumPy libraries to generate appropriate multivariate request pattern using temporal, spatial and redundancy distribution models. The generated workload has a generic structure as shown in Figure 3. Each line in the workload is generated as a JSON line and the total number of lines represents the total requests in a workload.

Location <lat, lon>	Request Size <bytes>	UID <64 bit>	Time <64 bit>	MIPS <64 bit>
{"location" : "40.712783,-74.00594" , "request_size" : 100 ,"uid" : 127 ,"time" : 0 ,"mips" : 100}				

Fig. 3: Structure of a generated synthetic access characteristics. Each line in the generated workload represents a line

IV. Validation

To validate Castnet, we use iFogSim and ns-3, as examples of already verified research simulators. We use the following two relevant metrics to establish the reliability of the results obtained from Castnet:

1. the response time trends for given edge nodes to validate the latency output of Castnet; and
2. the data transferred between the edge and cloud nodes over the simulation time period to validate the bandwidth consumption output of Castnet.

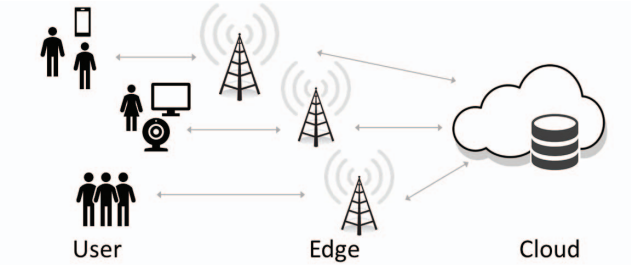


Fig. 4: Experimental setup used in Castnet validation.

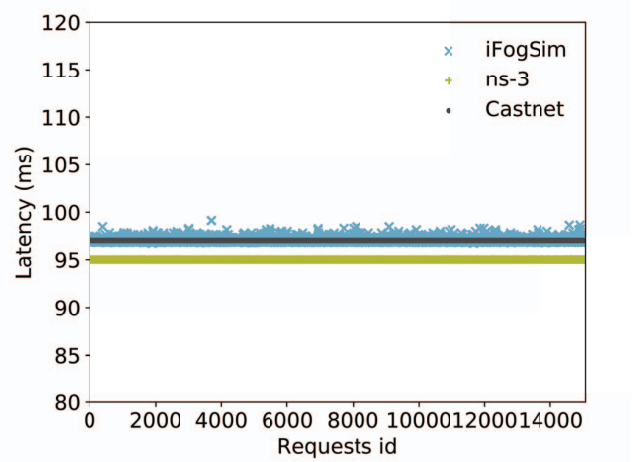


Fig. 5: Client response time trend for forwarding proxy server at the edge.

Workload. The workload used for these experiments consists of approximately 15k requests following a Gaussian distribution, analogous to one of the most common request pattern distribution pattern in a real world scenario [23], [24], with the simulation running over a span of 1 second. The focus of this exercise is not to quantify the merits but to compare the trends between different simulators and to validate Castnet. We ran two sets of validation experiments.

A. Experiment: Edge function as a simple proxy

Setup. We used the two-tier architecture with three edge nodes and one backend server as shown in Figure 4 where an edge node receives the request from the client and acts as a proxy server, forwarding it to the backend for further processing. The use of edge as a proxy server is the minimal functionality needed for an edge function. We use this experiment as baseline validation for the Castnet simulator.

Results. Figure 5 shows the time taken to serve each request in all three tools. When compared to the existing simulators, Castnet provides similar trend. However, ns-3 being a network simulator, does not include a compute model to account for

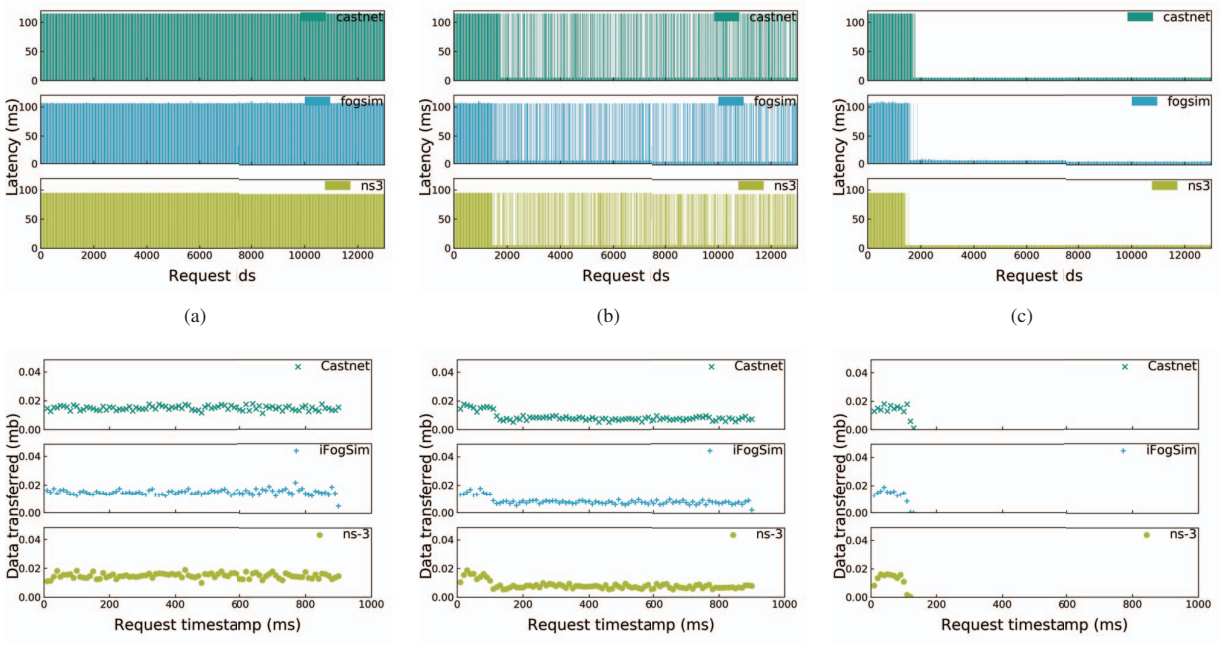


Fig. 6: Time taken by the application to serve the request and the data transferred between the edge and cloud server as the cache capacity on the edge server increases. The 6(a) and 6(d) depicts the case with no cache at edge, 6(b) and 6(e) represent the scenario when edge cache can hold 50% of the unique elements in the cache and finally, 6(c) and 6(f) have the cache capacity to hold all the unique elements in the workload.

any forwarding delay. This explains the difference in the absolute values between Castnet and ns-3. The oscillations observed by iFogSim are due to modeling network congestion effects. Currently Castnet uses a simple network model which does not take into account network congestion and hence provides similar response time for each of the requests.

B. Experiment: Edge function a content cache

To gain further confidence in the evaluations produced by Castnet, we ran a more complex scenario involving caching at the edge as a second validation experiment. The reason to choose caching is that in a real world scenario, it is the first intuitive use of edge nodes [25].

Setup. We again use the two tier architecture from the previous experiment and implement a caching edge function. In this case, the request made by the client is first served by the edge node where the edge application looks if the requested data is present it its cache. A cache hit results in the request to be served by the edge server, hence providing a lower response time and reducing the backhaul bandwidth usage. In case of a cache miss, the request is forwarded and served by the backend server. We implemented the same caching function and integrated it in all three simulation frameworks. We report on the effort required to do this in the following section.

Experiments. We conduct three experiments to understand the impact of caching at the edge, where each run involves a configuration with different storage capacity at the edge node. In the first experiment we do not configure any cache at the edge. 6(a) shows the expected latency values for the requests as they are served from the cloud. Next, the cache capacity is increased to 50% of the total unique elements present in the request workload. 6(b) and 6(e) depict the anticipated trend of the response time for all the requests over the simulation period and the amount of data transferred between the edge and the cloud server, respectively. For the final run, the cache capacity is increased to hold 100% of the total unique elements present in the request pattern. The early requests require more time as they are not cached at the edge, however, as the edge cache warms up, we observe a lower response time and no backhaul data transfer. 6(c) and 6(f) shows a similar trend across all the three simulators. Lastly, Table III shows that there is not much variation in the minimum, maximum and average time taken by the three simulators to serve the request.

Time taken (ms)	**iFogSim**	**ns-3**	**Castnet**
Minimum	5.01	5.34	5.00
Maximum	112.02	110.9	95.05
Average	17.80	16.32	13.63

TABLE III: Time taken by different simulators to serve the request when cache is placed on the edge server.

V. Evaluation

We next present results from the experimental evaluation of Castnet, comparing it with the iFogSim and ns-3 simulators. We seek to answer the following questions:

- How much effort is needed to use Castnet vs. the other available tools?

	Deterministic		Non-deterministic	
	Sparse	Dense	Sparse	Dense
Request range served by per server per sec	1	20	3-8	15-40
Total requests	1.5 M	1.5 M	1.4 M	1.6 M

TABLE IV: The workload distribution used in performance testing where the non-deterministic workload uses Gaussian distribution. (M = million)

- How fast is Castnet in getting to an insight vs. the other available tools for a given edge deployment configuration and a workload pattern?
- How does Castnet behave when used for different edge deployment scales and dense workload patterns vs. the other available tools?

A. Simulation Setup and Workload

All experiments are conducted on a quad-core Intel(R) Xeon(R) CPU X3430 @ 2.40GHz machine with 16 GB of RAM, i.e., a desktop class machine. We chose to use a desktop class machine intentionally, to highlight the lightweight nature of our tool. Due to lack of publicly available large-scale workloads, we use synthetic workloads generated by the Castnet workload generator. The workloads are chosen to represent different workloads for experimenting with up to 500 edge nodes, as shown in Table IV. Unless otherwise noted, we use the two-tier architecture with three edge and one backend server, shown in Figure 3.

B. Effort-to-Use

To demonstrate the ease of use of Castnet in developing and evaluating the trade offs for various edge functions, we developed caching at edge (discussed in IV) for all the three simulators. This required us to use the abstractions offered by each tools to represent the same scenario. Castnet and iFogSim consist of application level models such as data generation, fog devices, infrastructure and resource monitoring model. ns-3 is a network simulator and provides abstraction models to support network communication such as protocol stacks, peripheral cards and topology helpers. Ease of use can be subjective so we use lines of code (LOC) that need to be written using each tool as the metric for comparison. Table V shows the LOC needed to evaluate the caching edge function with each tool. We break down the experiment construction by the various models provided by the framework – infrastructure and placement, network topology, edge function logic, and helper functions. Castnet reduces the required LOC by 84% and 74% as compared iFogSim and ns-3, respectively. Further, we report that once written, it is much easier to change parameters in Castnet vs. the other tools, since it requires only changing values as opposed to changing code. *In summary, Castnet can be used with minimal effort from its users to gain insights quickly.*

C. Time-to-Insight

We compare the time-to-insight, or the time to complete a simulation and obtain answers, for Castnet, iFogSim and ns-3. We perform the experiments with each of the workload patterns illustrated in Figure 8, with parameters specified in Table IV.

Simulator	**iFogSim**	**ns-3**	**Castnet**
Infrastructure	400	345	66
Topology	160	105	10
Application	312	174	70
Miscellaneous	327	90	42
Total	1199	714	188

TABLE V: Average count of lines of code required to implement an edge application on different simulators.

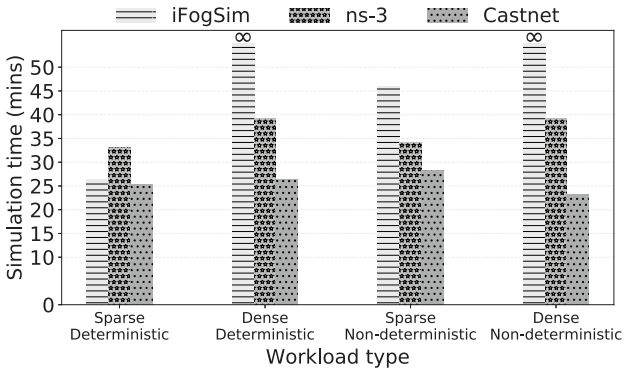


Fig. 7: Comparison of simulation time for an edge forwarding proxy application under different workload distribution. The ∞ denotes simulation taking a long time, which may even exceed beyond a day.

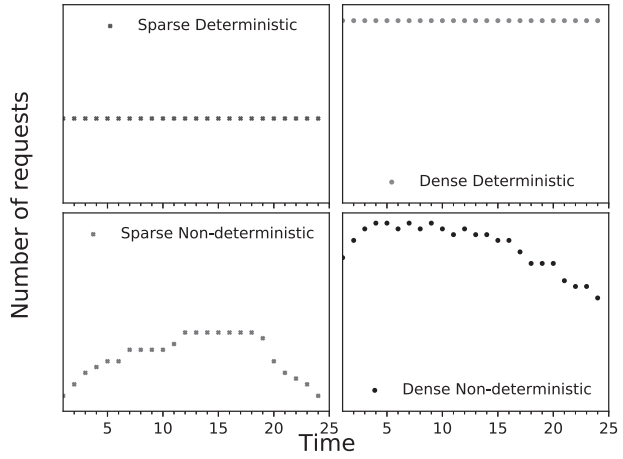


Fig. 8: Intuitive graphical representation of different workload categories.

Castnet outperforms iFogSim and ns-3 in all scenarios. Even with increase in the request rate served by each edge node, Castnet produces answers within ∼25 minutes, whereas iFogSim was unable to complete the simulation even after a long time, exceeding even beyond a day. For instance, for the deterministic dense distribution request model iFogSim was able to process only 60k out of 1.5M total request in a span of 12 hours, in contrast to Castnet, which was able to complete the entire simulation in 26.3 minutes. We were able to attribute the slow performance of iFogSim to its single event queue-based architecture, and the increased complexity

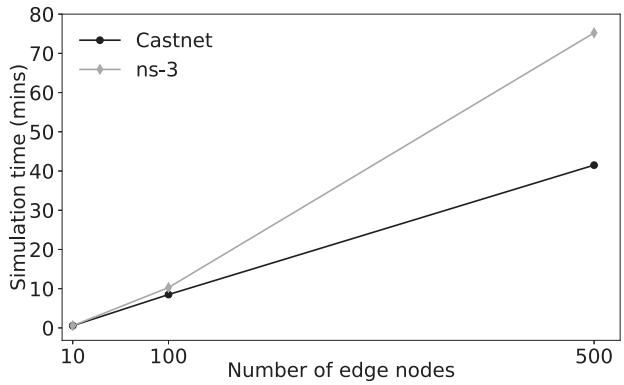


Fig. 9: Time required to complete the simulation with increase in number of edge node, keeping the total number of requests served by each node constant. iFogSim is not presented due to the unreasonably long completion time, which exceeds beyond a day, to finish the simulation for the given scale and workload.

of the resulting queue manipulation operations. Although ns-3 was able to complete the simulation with reasonable time (still 17.2-40.4% more than Castnet), it is still infeasible to use as an edge simulator without substantial modifications. This is because as a network simulator it does not provide appropriate abstractions required to incorporate into the simulation edge application logic and to account for its use of computational resources. *In summary, Castnet reduces the time-to-insights compared to the state-of-the-art available tools, to order for minutes for a realistic deployment and workload pattern.*

D. Scalability

To gauge the scalability of Castnet, we increase the number of edge nodes from 10, 100 to 500, and evaluate caching at edge, using the dense deterministic workload pattern, and experiments where 5 requests are served by each server per second, for a total of 1000 seconds. Figure 9 compares simulation time between Castnet and ns-3 only. We were unable to use iFogSim because it requires an unreasonably long time; we let it run for 24 hours without it being able to complete the simulation for the above mentioned workload. Castnet required 0.6, 8.5 and 41.5 minutes, respectively, to complete each simulation, compared to ns-3 whose completion time is 17.31% to 44.83% longer. *In summary, Castnet scales well with increasing number of edge deployments and is better than the state-of-the-art available tools.*

VI. Usage Scenarios

We present two concrete use cases to illustrate how Castnet can be used, and to highlight the extensibility of the framework. The first use case provides a more detailed description of the caching scenario used extensively in the earlier parts of the paper. The second one is based on an intrusion detection and prevention system [26].

A. Caching at edge (extension)

The experiment uses the two-tier architecture model used earlier. The infrastructure configuration of nodes and their placement is defined in the configuration file using infrastructure and placement attributes. The experiment uses workload consisting of total of 40k requests in a time period of 1 second, following a Gaussian distribution. Castnet's default reader supports JSON format however, the `play()` API exposed through the workload abstraction can be overridden to support other workload formats as well.

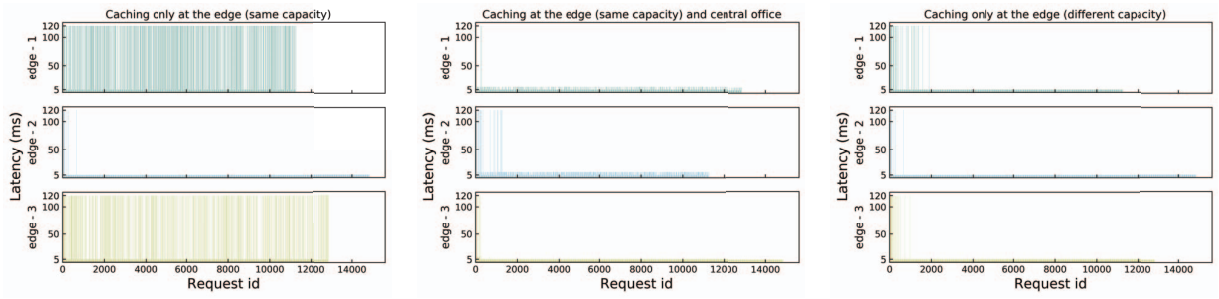
The first experiment performed consists of equal cache size at each edge location, capable of caching 150 unique requests. 10(a) shows the time taken by the requests served by different the edge locations. The trend obtained from Castnet shows that the existing size of the cache is suitable for edge node 2, however, there is a need to increase the cache capacity at edge nodes 1 and 3. To quickly evaluate few other infrastructure configurations, the infrastructure override option present in the configuration file, as shown in Listing 1, provides an easy way to test different cache capacities for edge nodes 1 and 3. 10(c) displays the reduced latency for the requests by providing the override values shown in Listing 2. Additionally, one might want to evaluate a scenario keeping minimum cache at the edge nodes along with a common cache at the central office to increase the overall performance. This changes the topology to a three-tier architecture model with edge nodes, central office and backend server. This change in topology requires only a change in the infrastructure attribute. In doing so, 10(b) shows the reduced latency for the requests, as estimated by Castnet. Thus, using the trends obtained from Castnet the content provider can quickly evaluate various infrastructure options and different topologies to determine the choice which can improve the service performance. As future work, we aim to provide an insight on the expected amount of compute and storage resources based on the request pattern served by the node.

```
INFRASTRUCTURE:
  USER:
    LOCATION: workload.json
    ...
  CELL_TOWER:
    DEFAULT_MIPS: 2000
    DEFAULT_BANDWIDTH:
      UPLINK: 100
      DOWNLINK: 100
    DEFAULT_LATENCY: 5
    CATEGORY: EDGE_CACHE
    CACHE:
      CACHE_TYPE: LRU
      CACHE_CAPACITY: 150
  ...
INFRASTRUCTURE_OVERRIDE: infra_override.json
...
```

Listing 1: An example configuration file snippet

```
{ "TYPE" : "CELL_TOWER",
  "LOCATION" : "33.68,-84.50",
  "CACHE_CAPACITY" : 260 }
{ "TYPE" : "CELL_TOWER",
  "LOCATION" : "34.68,-84.29",
  "CACHE_CAPACITY" : 450 }
```

Listing 2: An infrastructure override file snippet



(a) (b) (c)

Fig. 10: The latency for the requests with change in the cache capacity at edge based on the trend obtained from previous experiments.

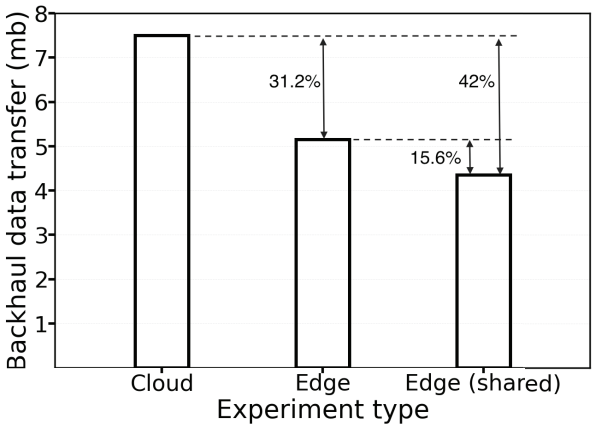


Fig. 11: The data transfer between the edge of the network and the data center when intrusion detection and prevention is done at the cloud, edge of the network without and with data sharing between the edges respectively.

B. Intrusion detection and prevention

Intrusion detection and prevention systems (IDPS) monitor a network in real time for malicious activity, access by unauthorized person or policy violations. These systems are classified into two categories: signature based detection, which uses patterns such as known instruction sequences or byte sequences in network traffic, and anomaly based detection, where the activity is classified based on a heuristics model or rules, rather than patterns or signatures, and the system attempts to detect if the request falls out of normal system operation.

We perform an experiment using Castnet which helps in understanding the benefits of deploying at an edge an intrusion detection system with anomaly-based detection. We evaluate two use-cases, one where the information between edge nodes are not shared and second where malicious IPs recorded at one edge is shared with the other nodes. For the purpose of our experiment, our IDPS model marks a request from an IP address as malicious if it gets more than 5 requests within a predefined amount of time (1 second). All future requests from the blocked IP addresses are then dropped automatically. Listing 3 demonstrates the ease of adding a custom edge function in Castnet by using the `pre_process_request()` handler exposed through the server abstraction.

A DDoS attack typically floods the service network with lots of requests and temporarily or indefinitely makes the server resources unavailable to its intended users. The deployment of IDPS at the edge could be an effective tool in handling such distributed attacks. It can contain the impact of such attacks to specific areas, ensuring a smooth experience to the rest of the users [26]. The experimental results obtained from Castnet for a workload of 150k request following a Gaussian distribution spread over an hour help in understanding the impact of moving such a model from the cloud to the edge of the network. Figure 11 illustrates a reduction in the backhaul data transfer by 31.2% when IDPS is moved to the edge. Further, Table VI demonstrates that data sharing among the edge helps in detecting malicious users more effectively, resulting in an overall 42% reduction in the backhaul data transfer for the given workload.

```
bool pre_process_request(DataEntry *data) {
  ...
  //pre process the data and check if it is a valid request
  if (is_blocked(data->get_id()) ||
     !is_valid_request(data->get_id())) {
    blocked_id.insert(data->get_id());
    data->set_compute_state(DROPPED);
    return false;
  }
  return true;
}
bool is_blocked(size_t data_id) {
  return (blocked_id.find(data_id) !=
      blocked_id.end());
}
bool is_valid_request(size_t data_id) {
  bool result = true;
  auto it = current_visit_count.find(data_id);

  //able to find the id and check if count is more than
       threshold drop the packet
  if (it != current_visit_count.end()) {
    current_visit_count[data_id] += 1;
    if (it->second > threshold_limit) {
      result = false;
    }
  } else {
    current_visit_count[data_id] = 1;
  }
  return result;
}
```

Listing 3: Castnet IDPS code file snippet

	Data center	Edge				Edge (with data sharing)			
	Cloud	Edge-1	Edge-2	Edge-3	Cloud	Edge-1	Edge-2	Edge-3	Cloud
Total requests	150000	51783	49228	48989	103172	51783	49228	48989	87002
Dropped requests	63002	16669	15313	14846	14	22150	20663	20185	4
Dropped requests %	42.0%	32.1%	31.1%	30.3%	0.01%	42.7%	41.9%	41.9%	0.004%

TABLE VI: The total number of requests served and dropped by each node with the change in deployment of IDPS from cloud to the edge.

VII. Conclusion and Future Work

In this paper, we presented Castnet, an open source simulation framework which allows for rapid evaluations of the trade-offs in MEC scenarios. The design of Castnet addresses the performance bottlenecks of existing simulators for MEC, thereby allowing for fast time-to-insight. In addition, it provides users with interfaces which permit easy exploration of different configurations of the MEC infrastructure, workload, and application deployment strategies, so as to understand the impact that changes in any of these dimensions can have on end-to-end performance or resource requirements. Castnet's modular design provides ample opportunities for future enhancements, such as through integration with more accurate network simulation models.

References

[1] Coetzee, Louis and Eksteen, Johan, "The Internet of Things-promise for the future? An introduction," 2011.

[2] Cisco VNI, "Cisco Visual Networking Index: Forecast and Methodology, 2016–2021," 2017.

[3] ETSI, "Multi-access Edge Computing," 2018, http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing.

[4] S. Srdjan, E. Theodoridis *et al.*, "SmartSantander: IoT experimentation over a smart city testbed," *Computer Networks*, vol. 61, 2014.

[5] G. Adjih, Cedric and Baccelli, Emmanuel and Fleury, Eric and Harter, Gaetan and Mitton, Nathalie and Noel, Thomas and Pissard-Gibollet, Roger and Saint-Marcel, Frederic and Schreiner, J. Vandaele *et al.*, "FIT IoT-LAB: A large scale open experimental IoT testbed," 2015.

[6] Riley, George F and Henderson, Thomas R, "The ns-3 network simulator," *Modeling and tools for network simulation*, 2010.

[7] Calheiros, Rodrigo N and Ranjan, Rajiv and Beloglazov, Anton and De Rose, César AF and Buyya, Rajkumar, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, 2011.

[8] Casanova, Henri, "Simgrid: A toolkit for the simulation of application scheduling," 2001.

[9] N. M, "iCanCloud: A flexible and scalable cloud infrastructure simulator," 2012.

[10] Gupta, Harshit and Vahid Dastjerdi, Amir and Ghosh, Soumya K and Buyya, Rajkumar, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, 2017.

[11] Ketan Bhardwaj and Ming-Wei Shih and Pragya Agarwal and Ada Gavrilovska and Taesoo Kim and Karsten Schwan, "Fast, Scalable and Secure Onloading of Edge Functions Using AirBox," in *In Proceedings of the 1st IEEE/ACM Symposium on Edge Computing (SEC'16)*, Washington, DC, 2016.

[12] Gluhak, Alexander and Krco, Srdjan and Nati, Michele and Pfisterer, Dennis and Mitton, Nathalie and Razafindralambo, Tahiry, "A survey on facilities for experimental internet of things research," *IEEE Communications Magazine*, vol. 49, no. 11, 2011.

[13] Kliazovich, Dzmitry and Bouvry, Pascal and Khan, Samee Ullah, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, 2012.

[14] Hashemian, Raoufehsadat and Krishnamurthy, Diwakar and Arlitt, Martin, "Web workload generation challenges–an empirical investigation," *Software: Practice and Experience*, vol. 42, no. 5, 2012.

[15] Barford, Paul and Crovella, Mark, "Generating representative web workloads for network and server performance evaluation," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1. ACM, 1998.

[16] Pang, Zhibo and Chen, Qiang and Han, Weili and Zheng, Lirong, "Value-centric Design of the Internet-of-things Solution for Food Supply Chain: Value Creation, Sensor Portfolio and Information Fusion," *Information Systems Frontiers*, vol. 17, no. 2, 2015.

[17] Xu, Xiaoya and Zhong, Miao and Wan, Jiafu and Yi, Minglun and Gao, Tiancheng, "Health Monitoring and Management for Manufacturing Workers in Adverse Working Conditions," *J. Med. Syst.*, 2016.

[18] Ghose, Avik and Biswas, Provat and Bhaumik, Chirabrata and Sharma, Monika and Pal, Arpan and Jha, Abhinav, "Road condition monitoring and alert application: Using in-vehicle smartphone as internet-connected sensor," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. IEEE, 2012.

[19] Bravo, Yesnier and Ferrer, Javier and Luque, Gabriel and Alba, Enrique, "Smart Mobility by Optimizing the Traffic Lights: A New Tool for Traffic Control Centers," in *Proceedings of the First International Conference on Smart Cities - Volume 9704*, ser. Smart-CT 2016. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 147–156.

[20] Broxton, Tom and Interian, Yannet and Vaver, Jon and Wattenhofer, Mirjam, "Catching a viral video," *Journal of Intelligent Information Systems*, vol. 40, no. 2, 2013.

[21] Schling, Boris, *The Boost C++ Libraries*, 2011.

[22] Niels Lohmann, "JSON for Modern C++," https://github.com/nlohmann/json.

[23] Laterman, Michel and Arlitt, Martin and Williamson, Carey, "A campus-level view of Netflix and Twitch: Characterization and performance implications," in *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2017 International Symposium on*. IEEE, 2017.

[24] Atikoglu, Berk and Xu, Yuehai and Frachtenberg, Eitan and Jiang, Song and Paleczny, Mike, "Workload Analysis of a Large-scale Key-value Store," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12. ACM, 2012, pp. 53–64.

[25] Zhu, Jiang and Chan, Douglas S and Prabhu, Mythili Suryanarayana and Natarajan, Preethi and Hu, Hao and Bonomi, Flavio, "Improving web sites performance using edge servers in fog computing architecture," in *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*. IEEE, 2013.

[26] Ketan Bhardwaj and Joaquin Chung Miranda and Ada Gavrilovska, "Towards IoT-DDoS Prevention Using Edge Computing," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*, Boston, MA, 2018.