Incremental Deployment and Migration of Geo-DistributedSituation Awareness Applications in the Fog

Enrique Saurez, Kirak Hong, DaveLillethun, Umakishore RamachandranGeorgia Institute of TechnologyAtlanta, GA, USA

Beate OttenwälderyInstitute of Parallel and Distributed SystemsUniversity of Stuttgart, Stuttgart, Germany

In Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (pp. 258-269). ACM.

Motivation and Goal

- A shift from the Information Age to the IntelligenceAge
  - Situation awareness applications
  - Geo-distributed, latency-sensitive, data intensive, involve heavy-duty processing, run 24/7
- Cloud Computing —> Fog Computing
  - Computing model offered by the cloud platforms should be extended to the edge of the network and the resources should become location-aware
- Fog Computing has to be augmented with the right distributed programming model

Foglets

- Facilitates distributed programming across the resource continuum from the sensors to the cloud
  - Provide a high-level programming model that simplifies development on a large number of heterogeneous devices distributed over a wide area
  - Provide an execution environment that enables incremental and flexible provisioning of resources from the sensors to the cloud

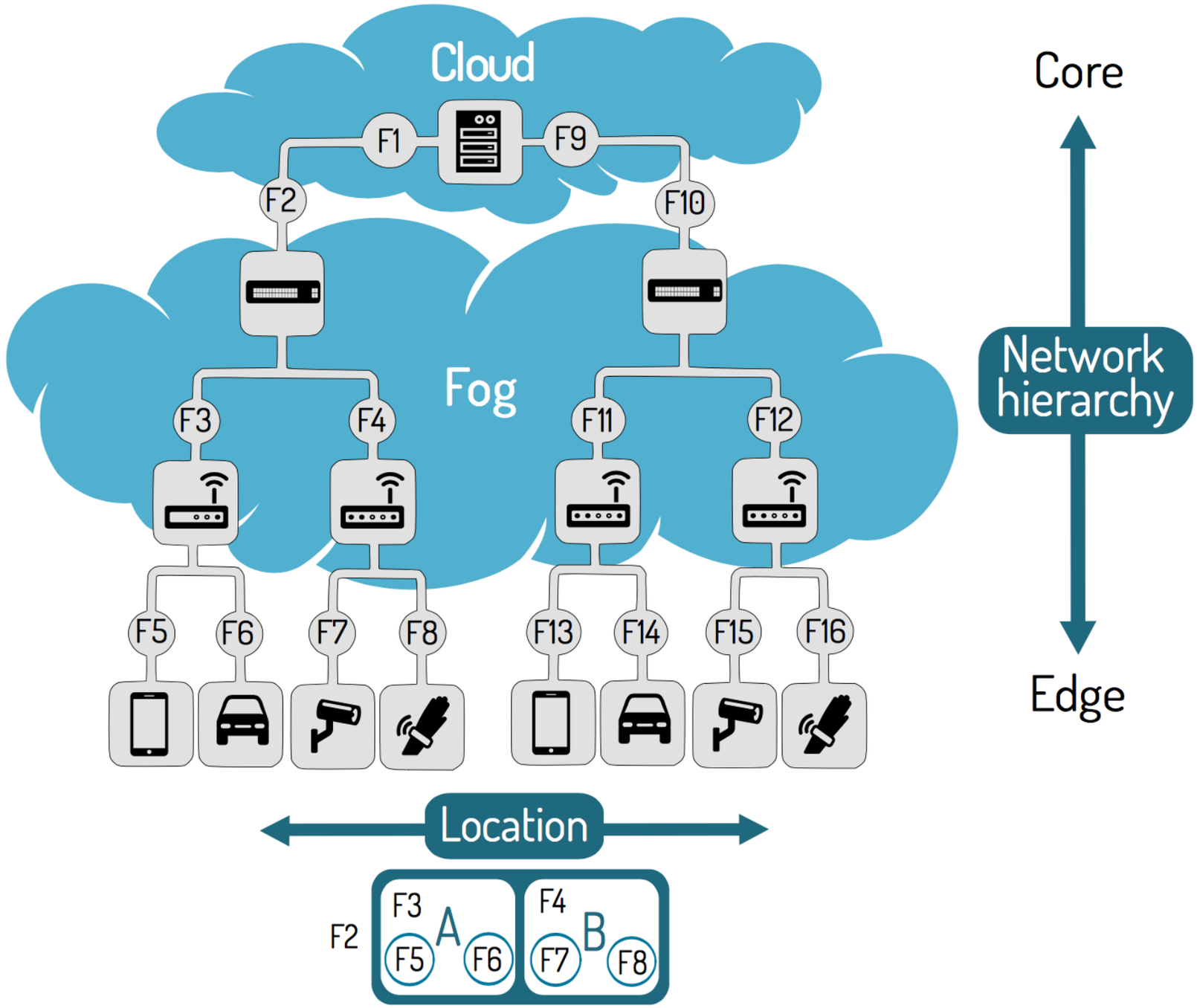
Four Main Functionalities

- Automatically discovers fog computing resources and deploys application components onto the fog computing resources commensurate with the latency requirements
- Supports multi-application collocation on any compute node.
- Provides communication APIs for components of the application to communicate with one another to exchange application state
- Supports both latency- and workload-driven resource adaptation and state migration over space (geographic)

System Assumptions

- A computational continuum that includes sensors and sensor platforms
- Physical devices are placed at different levels of the network hierarchy from the edge to the core network and associated with a certain geophysical location
- Physical devices called fog computing nodes are placed in the network infrastructure
- The fog provides a programming interface that allows managing on-demand computing instances
- Each computing instance has certain system resource capacities such as CPU speed, number of cores, memory size, and storage capacity

System Assumptions



Application Model

- Each process performs application- specific tasks such as sensing and aggregation with respect to its location and level in the network hierarchy
- A connection between two processes indicates a communication path allowed through the Foglets communication API
- Each Foglet process handles the workload for a certain geospatial region

API (communication)

- Foglets, application code consists of a set of event handlers that the application must implement and a set of functions that applications can call
  - Hierarchical communication API
  - Communication between application components
  - Stores application–specific data in a local object store called the spatio-temporal object store

Communication API

Interface	Description
void send up (message m)	Sends a message asynchronously from a child node to its parent node.
void send down (message m)	Sends a message asynchronously from a parent node to a child node.
void send_to (message m, node destination)	Sends a message to a specific destination node.
set<object> get(key k, location l, time t)	Get the application data that matches a key, location range, and time range.
void put object(object o, key k, location l, time t)	Put application data associated with a key, location, and time.

Table 1: Foglets API. The first three are communication primitives. The last two are for manipulating the local object store.

Handler	Description
void on_send_up (msg m)	Called when a new message arrives from a child node.
void on_send_down (msg m)	Called when a new message arrives from a parent node.
void on_receive_from (msg m, node source)	Called when a new message arrives from a peer node.

Table 2: Foglets Handlers. Application handlers that are invoked by the Foglets runtime on message arrival

API (applications)

- Once the code is written, an application developer compiles the code to generate a Foglet process image that can be deployed
  - Manage the application using the management interfaces provided by Foglets
  - Foglets uses several parameter to find appropriate computing resources for hosting application components (star_app)
    - appkey
    - Region
    - Level
    - Capacity
    - QoS,

Design Space Exploration

- Hosting Application Components
  - Virtual Machines
  - Containers
- Selection Strategies for Migration
  - Ensuring end-to-end latency restrictions and reducing the network utilization by planning the migration ahead of time
  - Using a Lyapunov optimization
  - Regarding the error bounds on the costs of hosting and migration

Foglets Runtime

- Discovery Server
  - Maintains a list of fog nodes available for hosting application components
- Docker Registry Server
  - Contains the binaries for the applications that have been launched on the Foglets infrastructure
- Entry Point Daemon
  - Started on each non-leaf fog node at system boot time
  - Awaits requests from the immediately lower level in the fog hierarchy to host a parent for a child
- Worker Process

Launching an Application

- Example
  - Surveillance application needing three levels of hierarchy
- 2-step Process
  - Writes the application logic for each of detector, face recognizer, and aggregator, as well as the handlers for each of the three levels
  - Creates the binary images for each of the application component, and registers the appkey and the images with the Docker registry server

Launching an Application

- Foglets runtime will ensure that Fog computing resources at the different levels are up in the region specified by the application
- Registry server will retrieve the detector process image from its key-value store and start up worker processes to host the detector on all the cameras in the region specified
- Neither the resources nor the worker processes for the upper-tiers are provisioned for this application at the time of launch

Discovery and Deployment Protocol

- Discovery
  - Finding the fog computing nodes (matching the capacity constraints) at the right level of the computational hierarchy
- Deployment
  - Spinning up a Docker container in a fog node to run a Worker process

Discovery and Deployment Protocol

- Child node executes a 2-phase join protocol to choose a parent from the list which obtained form Discovery server
- Each of the candidate parents sends back a response with their node-id (n) and state (s)

- READY - DEPLOYED (RD)
  - Join protocol is called
- READY (R)
  - Choose the best candidate node from the set of READY nodes to deploy a container hosting the application component
- BUSY (B)
  - Reinitialized with a bigger geo-graphical area

Join Protocol

- The child chooses the best candidate bc, which is geographically close to it to send a join message per the following equation:

$$\min_{\forall e \in W_{READY-D}} distance(e, child)$$

- Such an incremental application deployment ensures highly adaptive and elastic resource utilization driven by application dynamics and QoS needs from the edge of the network

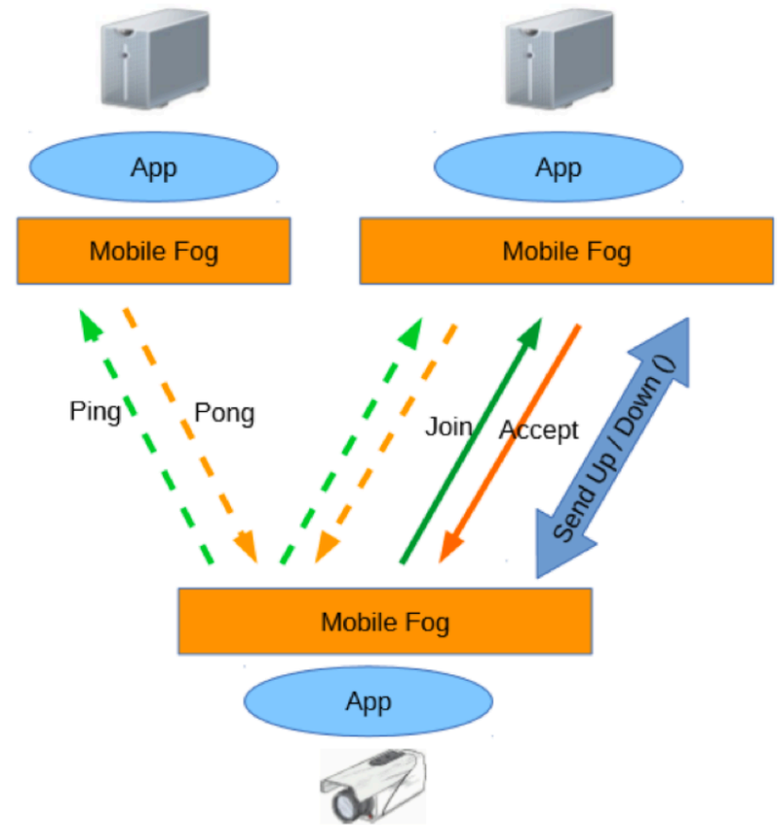


Figure 2: Join Protocol: The Discovery server gives a list of fog nodes in the upper-tier to the requesting child node. The protocol pictorially shown above results in the child choosing a parent to join from the list.

Migration

- Migration of an application component from one fog node to another (at the same level, usually the edge nodes close to the sensors)
- Two reasons
  - Meeting QoS expectations
  - Load balancing considerations
- Two aspects with respect to migration
  - Computation Migration (volatile state)
  - State Migration (persistent data)

Migration

- QoS-driven Migration
  - Proactive Migration
  - Reactive Migration
- Workload-driven Migration
  - Foglets runtime uses the stats provided by the EntryPoint daemon to make migration decisions if it finds the fog node is overloaded

Implementation Details

- Operating system Ubuntu 12.04
- Communication protocols are implemented using the ZMQ
- The Foglets implementation uses docker containers and RocksDB
- Emulate 16 fog nodes

Bringing up Foglets System

- The Foglets base Docker image (containing the runtime for the API calls, the communication libraries, and the Worker process to run the application) is built on top of the Ubuntu official docker image

Docker Image	Time (s)
Debian	8.6
Ubuntu	1.07
Foglets base	1.1
Foglets application w/base	18.36
Foglets application already deployed	0.42

Table 4: Startup times for different configurations of Docker images

Evaluation

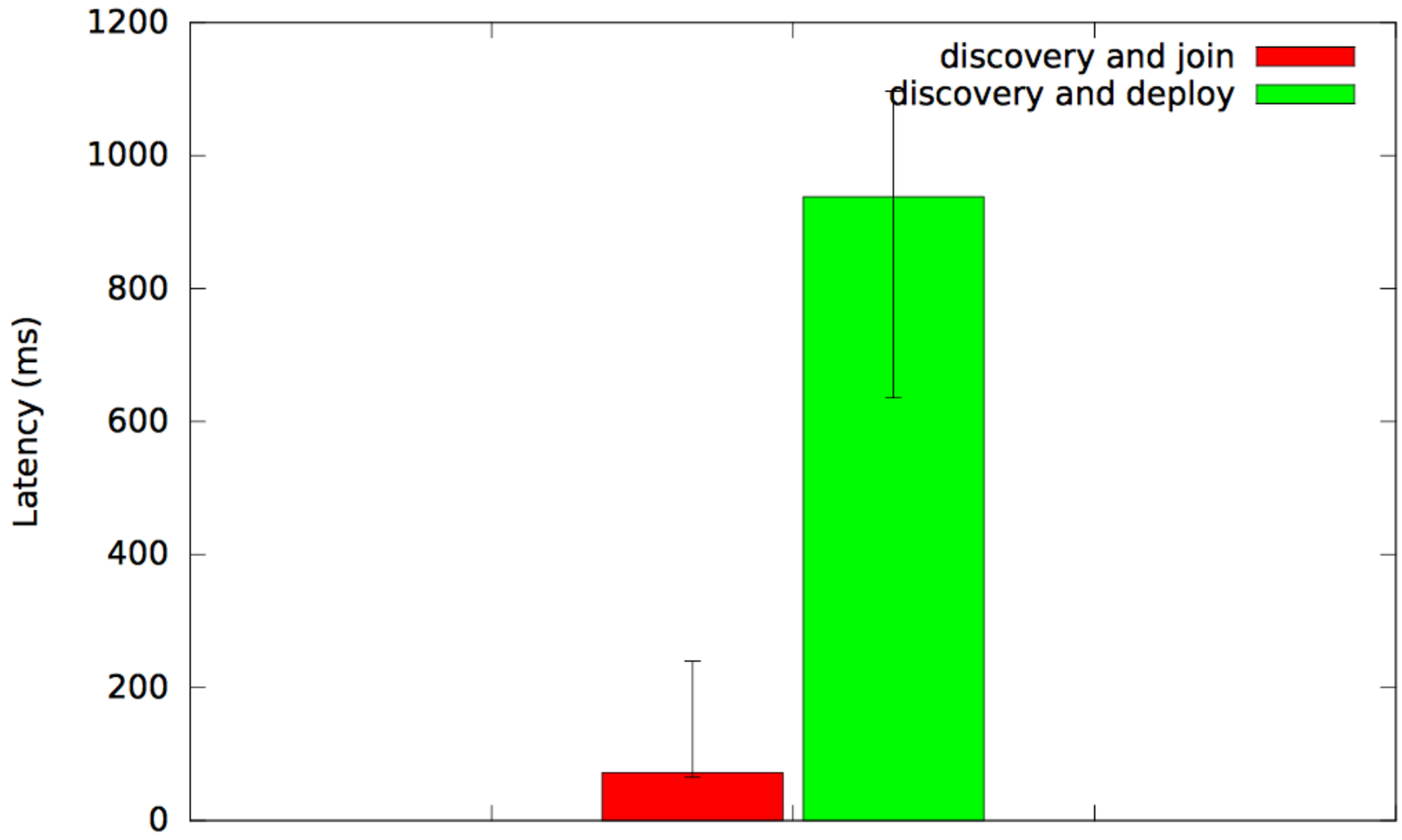


Figure 3: Comparison of Discovery-Join and Discovery-Deployment Operations. Error bars represent the 25% and 75% percentiles

Evaluation

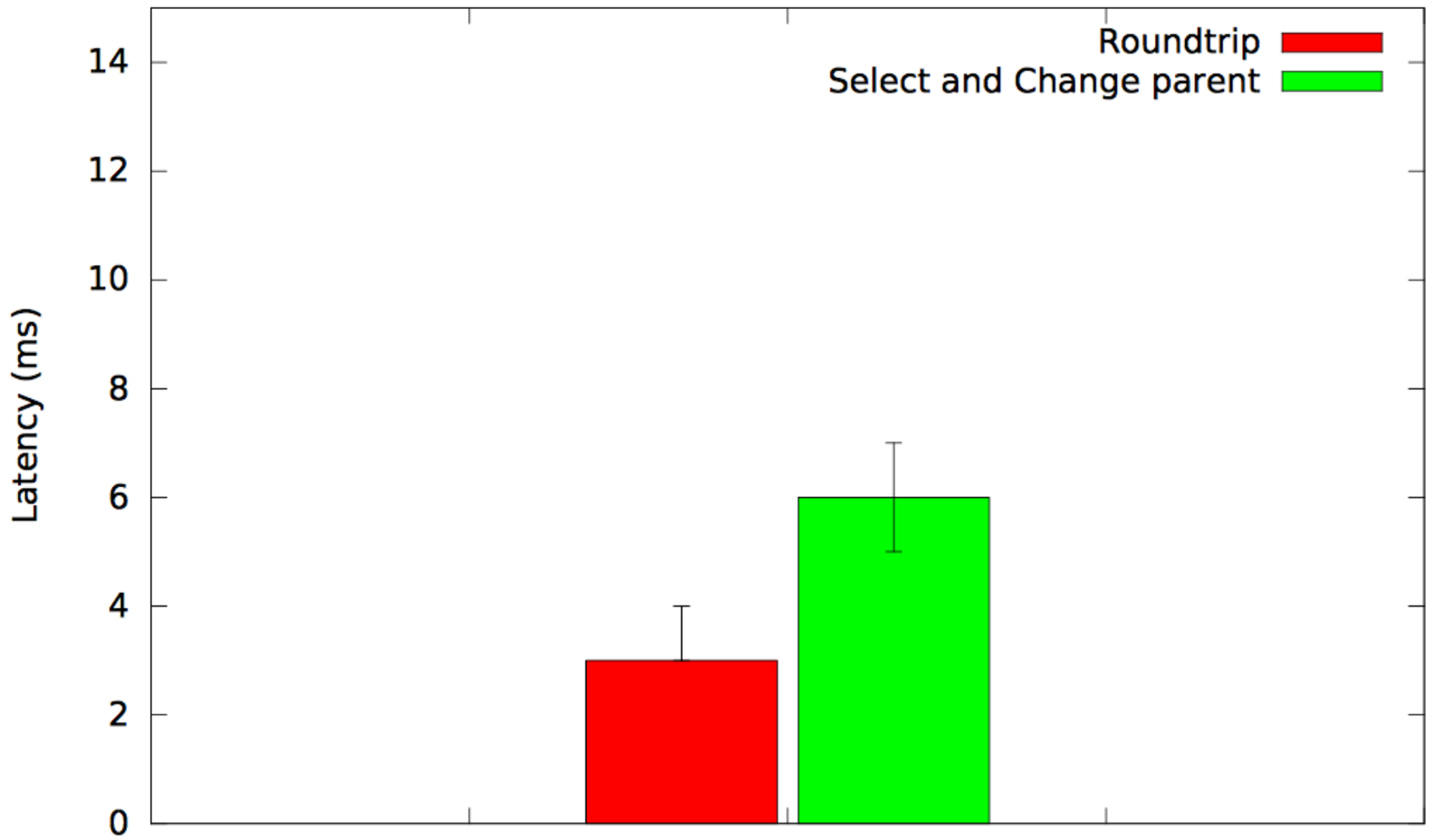


Figure 4: Proactive Migration Operation. As a point of comparison we show the network round-trip time. Error bars represent the 25% and 75% percentiles

Evaluation

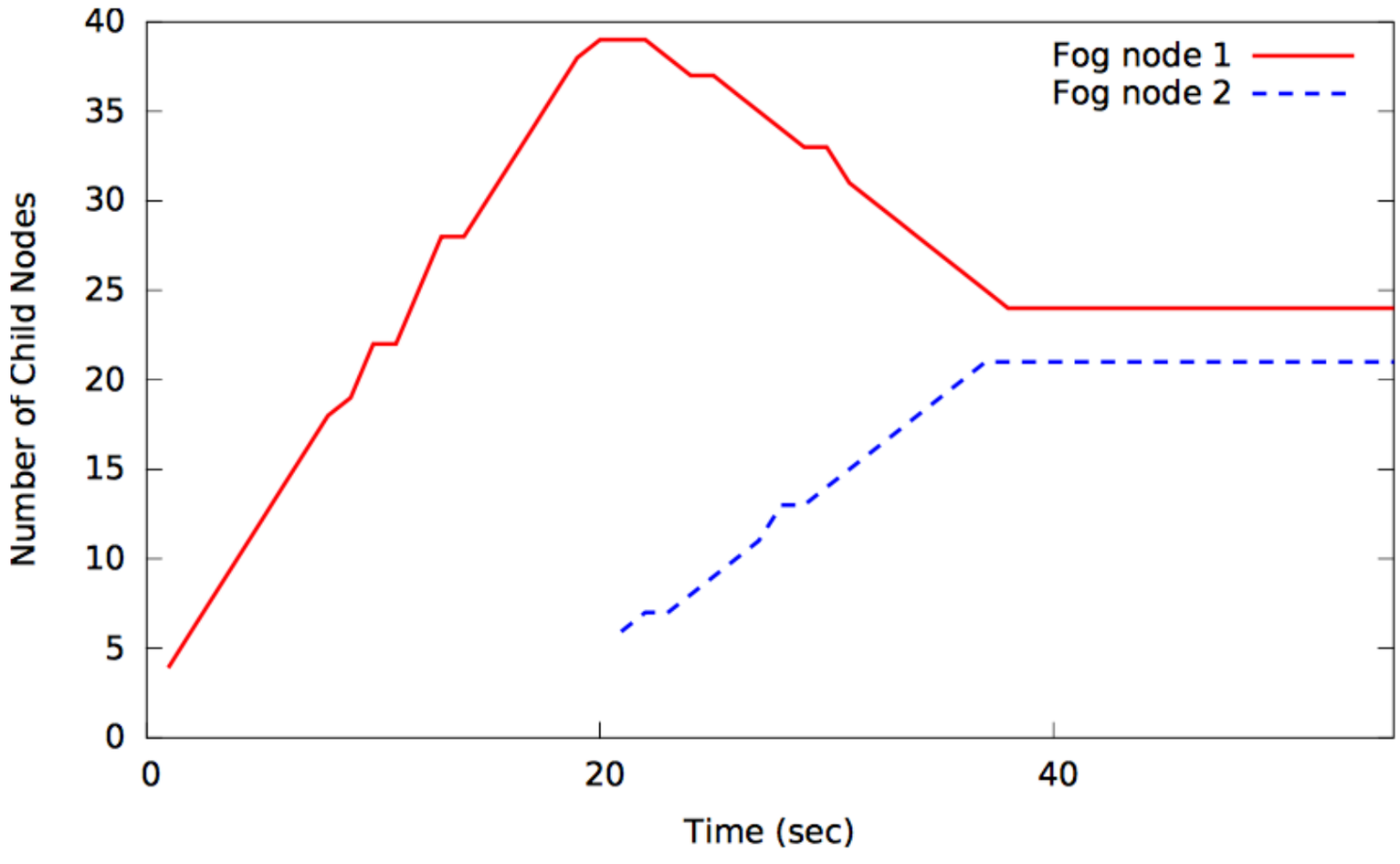


Figure 5: Workload Driven Migration. Over time fog node 2 accepts more children to offload the work from fog node 1.

Evaluation

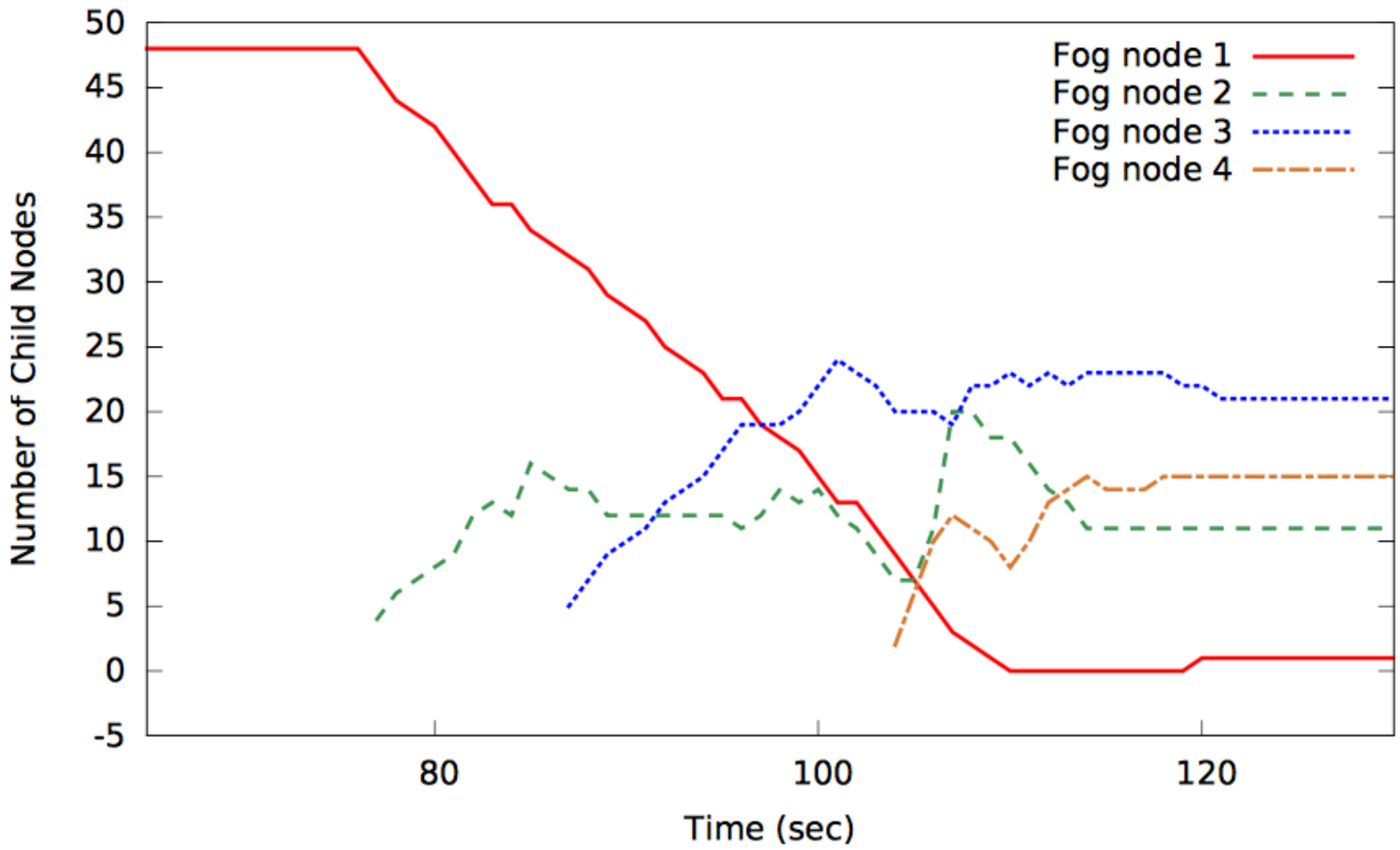


Figure 6: QoS-driven Proactive Migration. Over time the children are migrated to the fog nodes that are geo-local to the children moving in different directions.

Conclusion

- Situation awareness applications are emerging as important drivers of the IoT infrastructure
- Fog computing is a good utility computing model to cater to the edge computing needs of situational awareness applications
- Proposed a programming infrastructure for the computational continuum extending from the sensors to the cloud called Foglets