Comp Sci 4ZP6

Final Documentation

Team 1

*Andrew Azores – 1048083
Evan Holtrop – 1059591
Jazz Kersell – 1041571
Darren Kitamura - 0854359

Contents

Project Proposal: Suggested Project #5

Supervisor(s): Dr. Borzoo Bonakdarpour (CAS)

E-mail(s): borzoo@mcmaster.ca

Title: Distributed monitoring software for Android-based mobile devices

Description:

Runtime monitoring of distributed and networked applications is a challenging problem due to their inherent complex structure, caused by nondeterminism and occurrence of cyber and physical faults. However, given the recent explosion in using mobile devices such as smartphones and tablet computers and their applications, there is pressing need to have access to methods that can automatically monitor mobile applications in a network of devices to ensure their well-being. The goal of this project is to implement a distributed algorithm to monitor certain location-based properties of a network of Android devices. The algorithm does not assume a central monitor; i.e., it is fully decentralized in the sense that each Android device should be able to construct a consistent view of the global state of the network in a fully decentralized manner. This way, for a location-based service, each device is able to build an accurate shape of the network. The main application of this project is in autonomous vehicular networks. The project should deliver a prototype of the monitoring software. It has to be reconfigurable for different number of devices. A demo is also expected for different random scenarios of movement of mobile devices.

Skills required:

Java programming, experiences in Android OS a plus

Problem Statement

Description

Historically, networks of “processes” (devices) use some form of centralized server to maintain the state of the network in order to track a list of connected peers, facilitate discovery between peers, and marshal communications between peers. This approach is flawed, however, because it leaves the centralized server as a bottleneck or point of failure. If the network grows beyond the performance capabilities of the server, or if the server undergoes an external failure event such as a power outage, the entire network fails and must wait for the central server to reappear before the network can be re-established.

Decentralized networking is used in relatively few well-known networking protocols, the most well-known implementation may be the BitTorrent protocol. In this scheme, a centralized server is not strictly necessary (though one may be used in order to help increase the performance of the “swarm” in some scenarios), which removes the single point of failure problem. If any peer leaves the network for any reason, the network remains active and the remaining peers can continue to intercommunicate. This requires a different approach to communication and coordination between peers, however, since there is no “central authority” to report to or from which to discover any sort of global state. A necessary and unavoidable side-effect of this decentralized approach is an inherent concurrency in the network – it becomes much more difficult to determine when events occur in the network when there is no central authority tracking the events.

The goal of this project is to implement a real-time monitoring system over top of a decentralized networking stack which allows for each peer in the “swarm” to reliably determine their own current state as well as the state of each other peer in the swarm, thus allowing each peer to maintain its own copy of a best-estimate snapshot of what the global state of the network looks like, much like a centralized server would naturally be able to form. This real-time monitoring system has applications in, for example, swarms of flying drones, or self-driving cars. The technology will allow resilient and persistent networks of devices powering these vehicles to form and co-ordinate, for example allowing a network of self-driving cars to perform cooperative collision avoidance or route themselves along routes which will reduce traffic delays on average.

Requirements Specification

Change History

Version	Date	Author	Comments
0	October 12th, 2014	AA, EH, JK, DK	Initial version
1	January 8th, 2015	EH	Updating design document based on new project requirements
2	February 20th, 2015	EH	Adding new requirements
3	April 10th, 2015	EH	Updated Formatting
4	April 18th, 2015	DK	Fixed some grammar and content

Table 1 - Requirements Specification Change History

1. The Purpose of the Project

The User Business or Background of the Project Effort

- Implementing Dr. Bonakdarpour's algorithm that deals with distributed monitoring of a global state. To do this we will be keeping track of a global state of drone locations. Our overall goal is a physical working proof of concept for the monitoring algorithm.
- The business problem is strictly research at this point in time, but there are many applications if proven correct.

Goals of the Project

- Build an Android application that knows the global state of all drones in a swarm using the global state monitoring algorithm.
- The application must be able to recognize when location data is incorrect and correct accordingly.

2. The Stakeholders

The Client

- Dr. Bonakdarpour
- CMC Microsystems

The Customer

- Customers of the product will not actually be buying the android application we develop. They will be using the underlying monitoring algorithm for their own purpose.
- Vehicle manufacturers and companies that manage distributed systems.

The Hands-On Users of the Product

- Being a research project the hands-on users are limited to this Capstone group and our supervising professor.
- Upon a successful implementation and demonstration the hands-on users would include vehicle manufacturers, the military, and almost any organization using some sort of distributed system.

Priorities Assigned to Users

- Key users:
    - Developers
    - Supervising Professor
- Secondary Users:
    - Developers who use the end product algorithm for their own purposes

User Participation

- Supervising Professor expected to provide algorithm for doing distributed programming
- Users should not have to spend time dealing with our application, at most they will just have to enter in a few GPS coordinates and then send the drones on their way.

Maintenance Users and Service Technicians

- Supervising Professor / Graduate Students
- Development Team

3. Constraints

Solution Constraints

- Description: the network will be decentralized
- Rationale: by doing this it decreases start-up costs, and reduces the areas for potential failure within the network
- Fit Criterion: we will be using mobile devices that have direct communication, no server required.

Implementation Environment of the Current System

- Drones and Android Devices:
    - The app will run on android devices and control drones that the supervising professor will pick. The application will be designed to be easily moved to iOS and Windows phone if needed

Partner or Collaborative Applications

- As this is a research project there are no applicable collaborations at this time. When the project is more complete with a certain direction these opportunities may arise.

Off-the-Shelf Software

- Android Libraries:
    - Volley
    - Gson
    - Apache common I/O

    - Apache commons lang
    - Retro lambda
- Drone API:
    - 3DR services
    - droidplanner

Anticipated Workplace Environment

- McMaster University school grounds, open areas will be needed to test large drone swarms and permission will need to be acquired.

Schedule Constraints

- A deadline of April 2015 is a deadline for the overall project.

Budget Constraints

- Funding provided by CMC Microsystems

Enterprise Constraints

- A demo of the global monitor has been set by CMC Microsystems for February 27th
- Devices were shown using the multiple demo environments we had set up and the overall algorithm was explained to investors

4. Naming Conventions and Terminology

Glossary of All Terms, Including Acronyms, Used by Stakeholders Involved in the Project

- Supervising Professor
- Developers

5. Relevant Facts and Assumptions

Relevant Facts and Assumptions

- Self driving cars are gaining popularity due to Google
- Drones are being used in various countries to deliver packages
- Current distributed algorithms are not truly distributed, usually the monitoring is done by a centralized server or a leader node
- All devices on the network must be on the same network and subnet for interoperability

Existing systems can verify safety constraints, but not temporal constraints

6. The Scope of the Work

The Current Situation

- Centralized server will have to be replaced by a decentralized communication protocol
- Global state monitoring algorithm will have to be implemented across all android devices
- Scheme for continuous reliable data transfer between devices will have to be developed

The Context of the Work

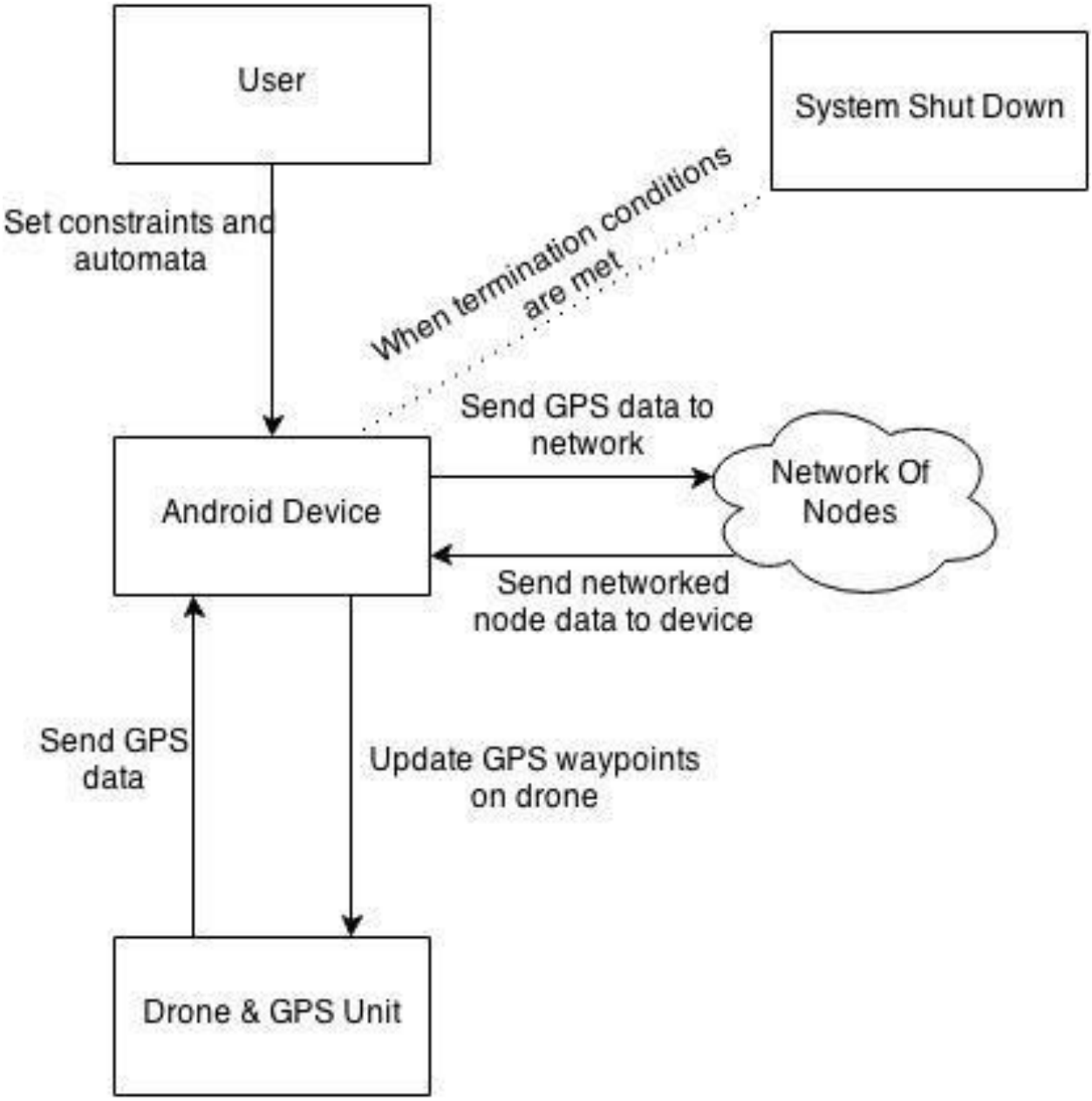

Figure 1 - Context of the Work Diagram

Work Partitioning

Business Event List

Event Name	Input and Output	Summary of BUC
Register with network	current node credentials (out)	Register with the network via NSD
Send Location data	Global State in (out)	Send location data of all known nodes upon request
Request Location data	Unknown Stat (out)	When more information about the global state is required send out a message to request the data
Receive Data	Global State (n)	After being requested another node will send the global state to the node requesting it
Incorrect Global State	Global State (out) correct flight path (out)	Using the global state the current node finds that it is in an incorrect state

Table 2 - Business Event List Table

7. The Scope of the Product

Product Boundary

- The product boundary of the application will be incredibly simple. The user would just input various GPS coordinates to a given node, once all drones are set they will perform actions as necessary until the global state is satisfied.

Product Use Cases

Name: Register with network
Trigger: service start
Precondition: device will have to be connected to local WiFi
Stakeholders: client, consumer
Actor: mobile device

Name: Send local state data
Trigger: request from another node
Precondition: another node has requisition for the global state
Stakeholders: client, consumer
Actor: mobile device

Name: request global state data

Trigger: current node doesn't have enough information to progress the global state
Precondition: current node doesn't have enough information
Stakeholders: client, consumer
Actor: mobile device

Name: Receive Data
Trigger: other node has responded to request data
Precondition: current global state cannot be progressed
Stakeholders: client, consumer
Actor: device

Name: Incorrect global state
Trigger: a drone has gone off track somewhere in the global state
Precondition: global state has been progressed to find that a node relevant to the current node is off track.
Stakeholders: client, consumer
Actor: Device / Drone

8. Functional Requirements

Requirement # 1	Type:	Event: 1
Description	The user must be able to enter in any number of GPS coordinates to a drone	
Rationale	The user should be able to send each individual drone to any place	
Fit Criterion	We will allow the user to input as many GPS coordinates as they want	
Customer Satisfaction: 5	Customer Dissatisfaction: 5	

Table 3 - Functional Requirements

9. Look and Feel Requirements

Appearance Requirements

- Basic user interface that provides the user feedback based on

10. Usability and Humanity Requirements

Ease of Use Requirements

- The application should work without the user needing to intervene

Personalization and Internationalization Requirements

- The application should be in English, a common language of all developers is English

Learning Requirements

- The user needs to understand LTL (Linear Temporal Logic) constraints, this is a fairly complex area of computer science. There are no reasonable learning requirements to set before that

Understandability and Politeness Requirements

- Most usability with the app will be high level computer science ideas

Accessibility Requirements

- N/A

11. Performance Requirements

Speed and Latency Requirements

- The network communications need to be fast enough so that the drones do not move onto the next GPS coordinate before the monitor returns a satisfied state
- The total time of the demo needs to be within 15 minutes. As this Is the max flight time of the Drones

Safety-Critical Requirements

- The drones are dangerous pieces of equipment and must be kept away from users, this is up to user discretion and is not something we can design into the application.

Precision or Accuracy Requirements

- The accuracy of our image recognition software needs to be able to recognize an object from whatever altitude the drone is flying.
- Our application needs to give the drone accurate GPS coordinates so that they will not move off track

Reliability and Availability Requirements

- Proof of concept requires the network should always be able to establish itself.
- Drones should get from beginning to end without diverging from the intended path

Robustness or Fault-Tolerance Requirements

The application should work with any reasonably sized drone swarm

- Reasonable will be defined when we have a better understanding of the packet size of the global state. As of right now we only have 4 devices to test with, this is far within the maximum number of nodes that can be used.

Capacity Requirements

- Application should fit and function on a hand-held device. Eg Nexus , iPhone

Scalability and Extensibility Requirements

- Application should be able to handle any sized drone swarm. Network should be able to handle this as well. We might need some stronger requirements once we figure out how large the global state is.

Longevity Requirements

- Application should function from monitor start until global state satisfaction or violation

12. Operational and Environmental Requirements

Expected Physical Environment

- Application will be expected to work in an open field area that has strong GPS signal

Wider Environment Requirements

- N/A

Requirements for Interfacing with Adjacent Systems

- Application should reliably interface with other devices that are also running the same application

Productization Requirements

- N/A

Project Issues

Open Issues

- The implementation of the Monitoring algorithm will be a topic we need to discuss later modularly implementing overriding the drone autopilot.

14. Off-the-Shelf Solutions

- Available solutions all use centralized servers.

15. Tasks

- Implement monitoring algorithm
- Implement drone override protocol
- Implement a reliable network between devices
- Implement a service that scans a stream of video (coming from the android camera) determines whether or not a circular object is within the picture

16. Migration to the New Product

- NA

17. Risks

- Algorithm being implemented doesn't properly deal with all global states.
- The device network cannot handle small global states.

18. Costs

- NA

19. User Documentation and Training

- Documentation of the algorithm will be provided to all members of the development team.
- Documentation of how to use the application will be provided to all members of the development team.

20. Waiting Room

- The drones will need to be able to operate in less than ideal weather conditions. For the initial implementation and testing the first version will not be taking this into account.

21. Ideas for Solutions

- Using better high quality hardware to deal with the same use cases.

Test Plan

Change History

Version	Date	Author	Comments
0	March 20th 2015	AA, JK, EH, DK	Initial commit
1	April 10th	EH	Updating Dates, Fixed formatting

Table 4 - Test Plan Change History

Test Factors and Rationale

Factor	Rational
Reliability	For the main application/demo, the UAVs are potentially very dangerous if they lose control, therefore it is extremely critical for this software to be reliable. Should some error occur there must be a recovery mechanism. The algorithm itself will also really depend upon having a reliable distributed network and reliable connections and communications channels to its peers.
Continuity of Processing	While this software will not be designed to meet real time requirements there is little room for computation to slow or stop. If a device enters a failure state but fails to notify the rest of the network in a timely fashion, then the system may be operating in an unacceptable state for an extended period of time before the fault is detected.
Correctness	This software is a proof of concept for a research paper, therefore it is essential that the implementation be correct according to the specification. A correct implementation is necessary to adequately demonstrate that the research is sound.
Performance	For the main demo/application, the system must be functional for up to 10 UAVs. It does not need to be able to spontaneously add or remove UAVs from the swarm. The system must be able to monitor constraints that apply to individual UAVs but not global constraints. The performance of the research paper algorithm itself is largely dependent upon the computational complexity of the algorithm – the software should be implemented such that it matches the same computational complexity.

Table 5 - Test Factors and Rationale

Testing Method

- Testing will consist of unit tests and manual testing. Initially the main effort of testing will be unit testing. This is necessary because the system has very little user interaction which makes manual testing somewhat unnatural at lower levels. Unit tests will provide white box and black box test coverage, and will ensure that individual components are functioning correctly. Manual testing will be conducted mostly at a high level to verify the system as a whole. There will be some lower level manual testing to verify the Device Communication Module. Unit tests will also be written to verify the state monitoring algorithm; mock devices can be created with simulated state transitions, and conditions about the evaluation of the state monitoring algorithm can be tested.

Types of Testing

Manual

- Manual testing will be used mostly for high level testing of the whole system. This is due to the fact that UAVs by definition have minimal interaction with users. There are two aspects of testing that can be done manually: whole system testing, and testing of the Device Communication Module. Interim demonstration applications will also be constructed, which may also be used for high-level testing. For example, a demonstration in which users can enter values for variables in a mathematical expression on various devices connected to the distributed network, and the state monitoring algorithm on each device will monitor that certain constraints are always met. For example, a test case could include ensuring that the algorithm correctly asserts failure of a non-negativity constraint when an expression is entered which may in fact become negative.
- Whole system tests will consist of loading a set of constraints and corresponding monitor automata onto each drone, and observing the behavior produced. Testing the Device Communication Module will consist of attempting to initiate communication between multiple (up to 10) Android devices and ensuring that all devices are aware of all other devices and are able to open communications channels to each other device.

Regression

- Regression testing allows developers to verify that their changes have not produced new problems in the code. Since these tests should be done frequently it is most efficient to use automated testing for this. This project will use unit tests as our primary means of regression testing.

Functional

- Functional testing, also known as black box testing, is used to verify that a piece of software is correct according to its requirements. Functional testing will ensure that: low level components return correct results when given correct inputs, up to 10 devices can connect and communicate with each other, and the system as a whole can interpret and execute based on given conditions, which is dependent upon the state monitoring algorithm functioning correctly.

Recovery Testing

- For safety and economic reasons, losing control of a UAV is an unacceptable circumstance. Recovery testing is therefore necessary to ensure that any problems that the system encounters will not lead to disaster. Testing will be conducted by injecting error conditions and observing recovery as governed by the state monitoring algorithm.

Compliance Testing

- Compliance with the specifications set out in our requirements document will be observed will throughout the manual testing phase.

Test Cases

Test Cases	Scenario	Expected Behaviour	Negative Result
Inputting coordinates for drones to fly	The starting point of the project before the drones take flight	Drones will begin the pre-determined operation	Nothing, the mission will need to be re-entered
Global State Transfer between devices	Drones are in flight and are sending and receiving state data of other drones	Drones in the swarm communicate their states to each other	The drone is not able to transmit and receive the state data and will retry
Recognition of drones being out of parameters	The flight path of one or more drones leads to them flying out of the parameters	One or more of the offending drones will reposition itself in the swarm	New drones will be elected to have their position readjusted
Leader direction change	The swarm leader determines the route must be adjusted	A new flight vector is sent to the swarm	The swarm does not receive the new flight vector and continues on the original path
Drone being added to network	A new drone comes online and joins the swarm	All other drones query the status of the newly joined device to determine its state	The drone is not added and will attempt to re-join, if it is already in flight the drone will land
Drone leaving network	The drone's mission is complete and disconnects	The drone leaves the network	The drone shuts down anyway
Drone being dropped from the network	During flight a drone or drones are dropped from the network	Re-establish connection and continue on with the pre-determined mission	Abort the mission and attempt to land
Drone flight vector correction	During a flight, conditions change and all drones in the swarm need to be adjusted	Determine the course correction and issue all drones in the swarm the new location	All drones in the swarm should attempt to land
State monitoring algorithm evaluating mathematical expressions	Values or coefficients for variables in a mathematical expression are input manually on Android or iOS devices for the distributed state monitoring algorithm to evaluate	The state monitoring algorithm should correctly report whether the specified constraints are fulfilled, violated, or are indeterminate	

Table 6 - Test Cases

Test Schedule

Date	Summary
15 Sep 14	Manual testing of Device Communication Module started
10 Nov 14	Proof of concept test cases finished
14 Nov 14	Proof of concept build
11 Nov 14 – 28 Feb 15	Refine test cases according to changing requirements
15 Apr 22	Final build

Table 7 - Test Schedule

Design Documentation

Change History

Version	Date	Author	Comments
0	21 Oct 14	AA, JK, EH, DK	Initial commit
1	April 10th	EH	Fixed formatting updated with drone module and update network module

Table 8 - Design Documentation Change History

Introduction

Purpose

- The Purpose of this project is to provide a proof of concept for a paper provided by the supervising professor.
- We will also be implementing the algorithm over a distributed network.

Description

- The Purpose of this project is to provide a proof of concept for a paper provided by the supervising professor.
- We will also be implementing the algorithm over a distributed network.

Scope

- The scope of this project is to create an easy to use interface that will allow our professor to expand and develop other applications. We will also provide him with the working Drone demo that is explained above. To do this we will need to implement his Monitoring algorithm, as well as develop a distributed network among devices to send the data required by the algorithm.

Design Principles

Distributed Network

- A network that exists between devices with no server. This way devices communicate directly with each other instead of communicating with a server.
- The benefits of a distributed network mostly have to do with points of failure, in a distributed network is a user drops then the network still exists. While in a Centralized network if the server disconnects all users are disconnected from each other.

Threads

- Everything that can be put to a separate thread has been done so. Our entire application runs asynchronously from itself, from the Service to the Server each module runs on its own.

Modularity

- Each module can run on its own without the need for another. The monitor module never defines the type of any objects that the user wishes to send across. Same with the networking module. The application layer is completely unaware of what happens on the top layer.

Development Details

Language of Implementation

- Due to our end goal being a demo of Android devices flying drones we were heavily limited to Java for our development languages.
- We chose java over using cross platform solutions like Xamarin so that we could use native java libraries, and the increased support that google includes.

Supported Frameworks/API

- Lombok
- Volley
- NanoHTTP
- NSD
- Gson

Timeline of Critical Components

Component	Date
Global Monitor	January 19th 2015
Distributed Network	January 19th 2015
Drone Pilot	April 22nd 2015
IOS Implementation	N/A

Table 9 - Timeline of Critical Components

Component Overview

Network Module

Device Info

- Used to serialize device information. Data tracked is IP, port, and a DeviceLocation structure.
- This structure is used as a container to send data across the network
- Function: toString - converts itself to JSON.

Device Location

- Used to keep track of location data of a given device, data tracked is latitude, longitude, altitude, barometer pressure, speed, bearing, GPS accuracy, gravity and linear acceleration.
- This structure is used as a container to send data across the network

NetworkPeerIdentifier

- Used to keep track of all other peers on the network. Each device contains a list of these so that they know of all peers.

Payload Object

- A wrapper for the object you wish to send over the network. Other meta-data is kept here for communication purposes.
- Upon testing we found that the CapstoneService can filter data as needed. We will need to test with a large number of devices to find if this is truly redundant.

Capstone Service

- Background service that manages automatic peer discovery and both incoming and outgoing message queueing and relay. It keeps track of all device information required (barometer, gravity, IP, port), manages receiving tokens and other Monitor module messages and hands these messages off to the Monitor, and allows the Monitor to make remote requests to other Monitors using remote identifiers which are also provided and managed by the Capstone Service.
- Most calls in this Service provide functionality to (un)package, send, and receive data to and from both local and remote recipients, either other CapstoneService instances or Monitor instances. Functionality to retrieve and share data from local sensors is also exposed.

Capstone Server

- Handles receiving communications over the network and passes received messages to the local Capstone Service for routing.

Monitor Module

Event

- An event is what is sent to the monitor to register that something has changed in the underlying application. An event is generated on one of message send, message received, or internal state change. An event consists of an Event Type (Send, Receive, Internal), a Network Peer Identifier, a valuation, and a vector clock.

Token

- This class is used to represent the computation slicing. It carries with it the known state are the originating process and picks up the knowns state at the destination process. When returned it is used to update the local view of the global state on the originating process.

Vector Clock

- Represents the logical time of each process. The logical time of a process is represented by an integer which counts the events that occur in that process.

- Function: Compare – compares two vector clocks and returns the required comparison depending on the outcome. Equal if both clicks are the same, Bigger if lhs has a more recent event, Smaller if rhs has a more recent event, Concurrent if each has some more recent event
- Function: Merge – Merge two vector clocks so that the merged clock contains the most recent occurring events in each process represented by the vector clocks. This returns a merged clock such that the new clock contains the most recent events. Throws an exception if clocks are of different size

Valuation

- The values of the variables being monitored in the process.

Process State

- Represents the state of the process. The state consists of a Valuation and a Vector Clock.

Automation

- The automaton is a nondeterministic finite state machine which represents the LTL3 property which is being monitored. It consists of:
    - Automaton States
    - Automaton Transitions
- It is important to note that the automaton does not track its current state. This is done by the GlobalView since depending on what information is known about the global state, the automaton may need to be in a different state.

Automation State

- represents a state that the automaton can enter
- type represents whether the state is an accepting, rejecting, or undecided state

Automation Transition

- Moves the automaton from one transition to another state.
- Has a list of conjuncts which represent the predicate that labels the transition. There are only conjunctive predicates. Any disjunctive predicates are dealt with by splitting them into multiple transitions.

Conjunct

- Represents a Boolean expression that is used to evaluate the predicate labelling a transition.
- The expression is represented by a Boolean Expression Tree

Boolean Expression Tree

- Represents simple Boolean expressions (IE, x > 2, y == 5) and uses Valuations to evaluate these expressions and return the result.

Global View

- Tracks what is known about the global state of the system, and uses tokens to retrieve information from other processes to update its knowledge of the global state.

Drone Autopilot

Autopilot

- Uses the MissionProxy and MissionItemProxy modules provided by Droidplanner to create a flight path for the drone
- Flight path is provided by the main MapsActivity, the user picks a path for the drone to fly

MissionItemProxy / MissionProxy

- these are two functions used to abstract the MissionItem and Mission classes provided in the 3dr Services API

System Architecture

- Application -> Distributed Network -> Global Monitoring Algorithm
- When an application runs into a situation where it needs information from another device it sends a message to the Global Monitor. The global monitor will find the information necessary from the device needed and update the global state with the new information.
- The distributed network is a layer in between the Application and the Global Monitor. The network is a link layer between the two that manages all IP/port connections.
- In our case the application layer will be the drone pilot program. Due to the fact that we do not have drones design of this module has been postponed.

Communication Protocol

- Communication is handled by using Json. All data is serialized into Structures described above. The structure then be converted into Json and are then sent over the network. The receiving device unpacks the Json and interprets the data as needed.

Monitor Algorithm Overview

- The monitor algorithm verifies that specified properties hold throughout the system. It receives events from the process it is monitoring and records the information from that event. When information from another process is needed the monitor sends a token to that process to request the information. The receiving monitor then processes the token, adding to it the requested information and sends it back to where it originated. This mechanism propagates information about the state of the system throughout the processes, and allows them to collectively determine whether the systems satisfies a given property.

Distributed State Monitoring User Guide

Change History

Version	Date	Author	Comments
0	27 Feb, 2015	DK	Initial commit
1	April 11th	DK, EH	Updated information and formatting

Table 10 - User Guide Change History

Legal and Copyright Information

Legal

- By usage of this software on you agree that McMaster's Computing and Software department and the developers are not in anyway liable for any damage on hardware or persons.

Copyright

- Reproduction of this documentation or software is prohibited unless written consent is received from McMaster University and the supervising professor. Copyright McMaster University 2015.

Introduction

Autonomous state monitoring is a practical Android based implementation of a theoretical algorithm. Using various Android devices on a network one can interact with system and ensure the conditions for the global state set are either satisfied or violated. The use of the Android platform allows for expansion into other things such as interfacing with drones or other peripherals.

Definitions

- LTL – Linear Temporal Logic
- NFC – Near Field Communication

Installation

Android

- There are a few steps to make sure that this application runs correctly from the app side.
- Transfer the pre-compiled APK to the device and install it on the system.

- Transfer the pre-generated JSON automata file and a Conjunct Mapping file. For more instruction on these files please refer to the PC based instructions.
- Ensure all devices are on the same network and subnet (this will usually be the case with most routers).
- Launch the application.

PC (Linux)

- The preferred Linux distribution for this setup is Arch-Linux which can be downloaded for free from the Arch website (https://www.archlinux.org/).
- Install SPOT which can be found in the package manager as “spot”, and if you wish to be able to view the automata graphically install GraphViz (optional)
- Install Ruby
- Use the bash script included in the Git repository under /ltl-testing and Ruby script under /tools/convert_automaton for generating the automata as below:
- ./formula_to_automaton.sh "Your LTL Formula here" | ruby automaton_to_json.rb > automaton.json

- Create an initial_state.json file to set the initial state of each device. Example format below:

```
{
	"^o":"InitialState",
	"valuations":[
	{
		"^o":"valuation",
		"variables":[
		{
			"variable":"x1",
			"value":"0.0"
		}]
	},
{
	"^o":"valuation",
	"variables":[
	{
		"variable":"x2",
		"value":"0.0"
	}]
},
{
	"^o":"valuation",
	"variables":[
	{
		"variable":"x3",
		"value":"0.0"
	}]
},
{
	"^o":"valuation",
	"variables":[
	{
		"variable":"x4",
		"value":"0.0"
	}]
	}]
}
```

- Create a file called numPeers that contains only the number of devices that will be on the network, for example for 4 devices the file would only contain the number 4 in it.
- Create a conjunct mapping file called conjunct_mapping.my the example
- contents are as follows:

```
#conjunct_name,owner_process,expression
A,1,x1 == 0.0
!A,1,x1 != 0.0
B,2,x2 == 0.0
!B,2,x2 != 0.0
C,3,x3 == 1.0
!C,3,x3 != 1.0
D,4,x4 == 1.0
!D,4,x4 != 1.0
```

- Load all of the files on the device into the monitorInit found on the Android devices sdcard folder
- Launch the Android application

Using the Android Application

Upon transferring files from the PC to the Android device, launch the application. When opening at the top you will be shown a list of other devices on the network that have detected each other and below a list of different options on how to trigger the device variables. At the top of each view the device will display the current variable and if the formula is satisfied or not.

Vision

- This view will open up the device's camera and start tracking objects that are a circle. When the device finds a circle every other device on the network will be made aware of a state change.

NFC

- This view will use the device's NFC reader and chips. Using a predetermined list of NFC uuid's the device can respond positively or negatively depending on if the tag was expected.

Cube

- This view generates a 3d model that rotates as the device is turned up. When the device reaches a 90 degree vertical the model will turn green and the variable on the device will update.

Drone

- This view shows a maps Activity that can be used to plot out a drone flight path. Once the path is chosen a user can connect to a drone and arm it, the drone will then fly along the path stopping at all red dots.

FAQ

- The app is crashing or complaining of missing files.
    - Make sure the automaton.json, initial_state.json, numPeers, and conjunct_mapping.my are all loaded into the app's folder.

- My device doesn't see any others that are running the app.
    - Ensure all the devices are on the same network. Some networks partition devices off in different subnets and if that is the case try setting one device to a hotspot and have all of them join that.

Test Report

1. Introduction

Purpose of the document

This section provides a brief introduction to the testing report for the Capstone project: Borzoo's Distributed State Monitoring Algorithm.

Scope of the testing

The scope of testing on this project is mostly based on user testing due to the high number of moving parts in this algorithm. An example of such would be functions to control the drone ie: travel forward 1 meter. Checking this programmatically is impossible because while the code could be correct and valid, the drone must still be observed.

2. Module Testing

Application Initialization

Purpose

Verify that the program has all the required files correctly formatted and installed correctly.

Sample Files

1) "firstSanity" folder files: automaton.json, automaton.my, conjunct_mapping.my, initial_state.json, and numPeers – Test cases that have proven to work
2) Missing files (none of the above included)

Test Cases

Application startup containing sanity check files
Application start up missing initialization files

Network Connectivity

Purpose

Network connective is absolutely vital to the system functioning correctly.

Sample Files

Multiple devices are connected to the same network and are on the **same subnet**.

Test Case

Network connectivity.

Cube Activity

Purpose

Ensure the activity in the application operates correctly.

Sample Files
N/A

Test Case
Rotate the device to be standing vertical and ensure the object on screen changes colour.

NFC Activity

Purpose
Ensure the activity recognizes the NFC tag when tapped to the device.

Sample Files
NFC tag with the UUID registered on the device.

Test Case
Tap the NFC tag to the device and an toast notification appears informing the user that it was accepted.

Vision Activity

Purpose
The vision activity is one of the processes that manipulate the device state in the algorithm.

Sample Files
Any object in the real world that is round for the camera to inspect.

Test Case
Use the device camera to look for round objects such as a watch face.

3. Module Tests

Summary of performed tests

Test No.	Test Case	Initial State	Input	Expected Output	Test Result	Test Conclusion
1	Application Initialization	Device loaded with sanity test files	N/A	Application loads without error	Behaved as expected	Pass
2	Application Initialization	Device is missing required config files	N/A	Application immediately crashes	Behaved as expected (crashing) but reason for crashing is not consistent	Pass
3	Network Conductivity	Devices are connected to the same network and	N/A	Each device recognizes the other devices	Behaved as expected	Pass

		are part of the same subnet		on the network		
4	Cube Activity	Device is held in any angle other than being held perpendicular to the ground	Device is held vertical and perpendicular to the ground	Rectangle on screen turns green	Behaved as expected	Pass
5	NFC Activity	Device has activity running with no NFC tag near it	NFC tag is read by the device	Device notifies the user that the NFC tag was accepted	Behaved as expected	Pass
6	Vision Activity	Device has activity running with no round objects by the camera	Hold a round object to the camera	Device recognizes the round object and reports the acknowledgement to the user	Behaved as expected	Pass

Table 11 - Summary of Performed Tests

4. Performance Testing

Automated Testing

Due to the size of the project and time allotted we were unable to take advantage of any automated test suites.

5. Usability Testing

Usability testing is performed to determine how effective user interaction takes place on the application. Due to the nature of the project being a more research-based implementation of an academic paper any sort of design and usability were not taken into consideration. When the paper has a more concrete final version released the team can sit down and design a proper user interface for the project.

List of Figures

List of Tables

Contribution

Andrew Azores - 30%

- Developed decentralized communications layer, defined interfaces and implemented facilities for communications between Service/Server, Monitor, and Monitored Processes.
- Provided technical experience with Java.
- Developed vision processing demo application (recognizes circle in view, counts occurrences of circles coming into view, reports count to monitor)
- Implemented asynchronous communications and processing wherever deemed feasible in the project
- Wrote Documentation

Evan Holtrop - 17.5%

- Wrote Documentation
- Developed Autopilot module
- Developed Application layer for Demos

Jazz Kersell - 35%

- Developed the Global State Monitoring Algorithm
- Wrote Documentation

Darren Kitamura - 17.5%

- Wrote Documentation
- Attempted porting all Android code to iOS
- Developed Autopilot Module
- Wrote Presentation Slides