國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

智能M2M系統的容錯研究

Research on Fault Tolerance for Intelligent M2M Systems

蘇適

Penn H. Su

指導教授：許永真 博士

Advisor: Jane Yung-Jen Hsu, Ph.D.

中華民國 102 年 3 月

March, 2013

國立臺灣大學碩士學位論文
口試委員會審定書

智能 M2M 中介軟體的容錯系統研究

Research on Fault Tolerant M2M Systems

本論文係蘇適君（學號 R99922157）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 102 年 3 月 14 日承下列考試委員審查通過及口試及格，特此證明

口試委員：許永真

（指導教授）

系主任 許永真

Acknowledgments

I would like to express my greatest gratitude to the people who have helped and supported me throughout my graduate studies. I am grateful to my advisor Prof. Jane Yung-Jen Hsu for her continuous support, from initial advice, contacts in the early stages of conceptual inception, through ongoing advice and encouragement to this day. I am also grateful to my co-advisor Prof. Kwei-Jay Lin for his undivided attention to details, ongoing advice and encouragement.

To Dr. Yu-Chung Wang and Prof. Chi-Sheng Shih, for their patience and their kindness for answering my many requests.

To Niels Reijers, my colleague who helped me in completing the project with his vast knowledge, black VM magic tricks, ideas, and thoughts that made this journey a lot smoother.

To my colleagues and my friends, thanks for the support and fun. I want to thank iAgent group, George, Jya-Cheng, Joey, Leeheng, Farmer, Bo-Lung, and Sio, who have directly or indirectly help out this effort.

I wish to thank my parents for their undivided support and interest who inspired me and encouraged me to go my own way to this day, without whom I would be unable to complete my graduate studies.

中文摘要

對於具有”部署一次，永遠運行”概念的物聯網而言，容錯移轉是這類分散式服務導向網路的必備條件。當設備更換時或系統出狀況時，必須利用資源再規畫去達成容錯移轉的機制。系統在運作時，異質性的或多工性的設備之間若不僅是端對端的訊息傳輸時，不管是設備或是訊息的複製都是昂貴且累贅。特別是當設備某種服務故障時，可由另外一個有能力提供相同服務的同級設備接替其服務，而不一定要由相同設備取代。利用長帶來記錄一連串複製的服務訊息，每一個同級設備都保存一致的長帶記錄。結合常用於失敗偵測的心跳協定，系統由異常回復的機制可以藉由操控分析分散狀態的長帶來達成。使用Arduino mega 2560相容設備所做的實驗結果顯示，我們已經能夠使小型網路系統故障復原，較大的網路實驗則正在進行中。未來研究方向包括確認網路的可擴展性，網路磁碟分割處理以及解決同步故障的問題。

Abstract

Failover for service-oriented distributed networks is a prerequisite to enabling Internet-of-Things (IoT) in the sense of "deploy-once, run forever." Resource reconfiguration is required to achieve failover mechanisms upon replacement of devices or failure of services. It can be particularly challenging when services in applications have more than end-to-end transmissions between devices that are heterogeneous or versatile, for which duplications can be costly and redundant. Specifically, a device with a failed service shall be taken over by another service peer, instead of a device counterpart to recover application as a whole. Strip is introduced to store a list of duplicated services, and, each service peer maintains a consistent view of strips. In combination with the heartbeat protocol which was widely applied for failure detection, recovery from failure can be achieved by manipulating strips in a distributed manner. Experiments using Arduino Mega 2560 compatible devices show that our approach is capable of failover in small networks, whereas experiments in larger networks are underway. Future research directions include addressing the scalability issue, network partitions and tackling simultaneous failures.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Internet-of-Things (IoT), also known as machine-to-machine (M2M), refers to the pervasive presence of uniquely identifiable objects (things) around us - such as Radio Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, which would be able to interact with each other forming a network like structure. The paradigm gradually leads to all objects surrounding us to become connected to the Internet, to interact with other connected objects on the other end of the web and have access to comparative information which would not be possible before. An example would be environmental monitoring where sensors are scattered across a vast land, where they could detect, identify phenomena at the area when it is happening.

IoT is being realized in M2M systems such as wireless sensor networks (WSN). WSNs are areas filled with network of tiny, resource limited sensors communicating wirelessly. Each sensor is capable of sensing the environment in its proximity. Wireless sensor networks are typically employed in a variety of applications ranging from home automation to military. Sensor networks offer the ability to monitor real-world phenomena in detail and at large scale by embedding devices into the environment. Deployment is about setting up an sensor network in a real-world environment, and it is a labor-intensive and cumbersome task since environmental influences or loose program logic in code might

trigger bugs or sensor failures that degrade performance in any way that has not been observed during pre-deployment testing in the lab. The real world has strong influences of the function of a sensor network that could change the quality of wireless communication links, and by putting extreme physical strains on sensor nodes. Laboratory testbed or simulator can only model to a very limited extent of those influences. There have been several reports on sensor network installations where they encountered problems during their deployment[1, 2, 3, 4, 5, 6, 7, 8]. Testbed in laboratory environment can still not model the full extents of the influences a real world environment could do. Deployment still a big problem in wireless sensor network applications. This is exacerbated by the fact that Wireless Sensor Network (WSN) programming can only be performed by experts because of resource constraints and their high failure probability. Hence, transforming the paradigm of service-oriented architecture (SOA) to sensor networks becomes an important research in the last few years. [9, 10]

WuKong, a middleware for service-oriented M2M systems, has set out to allow programmers develop M2M applications with ease by introducing a new programming paradigm called flow based programming (FBP) to offer drag-and-drop software components and flow-chart like programs with automatic sensor identifications, node configuration, and system re-configuration. [11] However, if a number of sensors or services failed, the whole system would be brought down, therefore failover mechanism is especially crucial for enabling IoT in the sense that applications could ”deploy-once, run forever,” and where applications degrade gracefully under failures.

1.1 Goals

In this thesis, we study the fault tolerance capability in WuKong. Users of WuKong deploy applications written in the form of flow based programming. Application consists of a network of components where each component represents a service in the physical network. In WuKong, users are allowed to specify different user policies for the same application so that systems can show personalized behaviors. When an application is deployed, each component is mapped to a service in the network. However, partial failures could bring the whole system down since each service depends on each other to function.

The goal of this study is to provide failover for application components such that it would meet user requirements specified in user policy. Thus the system also needs to design a user friendly user policy for fault tolerance such that it is intuitive for users to specify quantitatively how strong the fault tolerance a component should have that could directly being mapped to the lower physical layer. It is also important that the deployed system would be able to detect, recover from failures, and reconfigure the system autonomously.

1.2 Challenges

The way WuKong presents and models resources and applications, along with user policy and requirements, makes an unique challenge: User requirements might vary drastically from one to another. System has to embrace a variety of top-level and low-level configurations so components would play along with each other and eventually produce a working failover architecture for the network. Encompassing a variety of devices with different form factors, memory storage, radio bandwidth is also a distinct characteristics

in IoT, therefore the system and algorithmic design has to work among nodes with different specifications. In addition, duplications can be costly and redundant when devices are heterogeneous or versatile, such that instead of replaced by a device counterpart, it would be taken over by another local service peers.

1.3 Related Work

Previous work on the problems have not considered failover architecture for applications where relationships among services are critical and for which they would heavily influenced by user policies. Furthermore, after every failure recovery, none explained how existing services could find replaced services in such environment to the best of our knowledge.

In WSNs, failover techniques for data replications, and service redundancy are typically adapt for applications only with predefined services, fixed user requirements and services with only end-to-end links: read/write to local data storage. One example of such work with local storage for replications is described in [12]. In this work, some distinguished storage nodes are specified by Hash functions to collect data of certain types. Redundancy is achieved by storing replicas directly on neighbors nodes. But applications are still restricted to end-to-end links, since there is only one type of links in the application. Having more than end-to-end links in a M2M system allows a higher degree of complexity in applications to meet the user requirements when services could build on top of other low-level services.

In [13], a dynamic replication approach for local data storage where replicas are randomly distributed within a predefined replication range influenced by the specific replica number and/or its density is presented. Another dynamic replication approach for local

data storage where replicas are selected based on the scoring system defined by several physical resource traits from self-inspection that could reactivate services based on already collected and generated data [14]. However, this approach does not consider applications with dynamic user requirements, services with more complex links.

1.4 Thesis Organization

Chapter 1 gives an introduction to the system goals and challenges and outline the approach used to solve the problem. Chapter 2 gives a brief background overview of the topic that this work based on. We start by describing IoT, WSNs, and ends with an overview of WuKong. Chapter 3 describes our system design for the fault tolerance primitive. In this chapter we give detail description of our method and algorithms, including user policy for fault tolerance, strips, and reconfigurable redundancy architecture. Chapter 4 presents the metrics we would use to measure system performance, which is followed by evaluation of the results. Finally, chapter 5 presents the conclusion of the work, list of contributions and future work.

Chapter 2

BACKGROUND

2.1 Internet-of-Things

2.1.1 What is Internet-of-Things

Internet-of-Things is a powerful paradigm in the scenario of modern wireless telecommunications. It refers to the pervasive presence of uniquely identifiable objects (things) around us - such as Radio Frequency IDentification (RFID) tags [15], sensors, actuators, mobile phones, which would be able to interact with each other forming a network like structure [16]. Example scenarios where IoT would shine: alarms clock goes off early if there is traffic; plants communicate to the sprinkler system when it's time for them to be watered; running shoes communicate time, speed and distance so that their wearers can compete in real time with people on the other side of the world; medicine containers tell your family members if you forget to take the medicine. The paradigm gradually leads to all objects surrounding us to become connected to the Internet, to interact with other connected objects on the other end of the web and have access to comparative information which would not be possible before.

2.1.2 The Evolution of Internet-of-Things

The figure 2.1 illustrates an evolution of Internet. Communication between computers already exists that made possible for two PCs to send messages across large geographical areas. With the Internet, all the computers connected to the Internet can talk to each other and now with mobile phones, the connections have become mobile. It would be unimaginable 20 years ago, when you have to communicate with someone far away in immediate urgency, you can open up your phone and make a call to someone far away from you and initiate a conversation. Nowadays even more so with 3G networks, even phones can connect to the Internet and be part of the network. Social media, such as email, blog, facebook, twitter, has been uniting people on the Internet up until now. With social media, we could go beyond physical limits with whom we can keep in direct contact. Internet-of-Things will further make possible for massive amount of objects to be online and to enable objects to access external information about themselves through external services in their vicinity. [17]

2.1.3 Potential Applications

Today more things than humans are connected to the Internet [18]. It is estimated things will have 24 billion devices connected to the Internet by 2020 [19]. In addition, the computation capabilities per device are increasing. Today's smart phones are about 15 times faster than CPUs used in 1979 Cray-1 [20].

Potential applications of this new emerging paradigm is skyrocketing as more sensor networks, actuators, have been more common as sensing, communication, and analytics have matured that are ubiquitously embedded into everyday objects surrounding every where we go. Some examples are highlighted below:

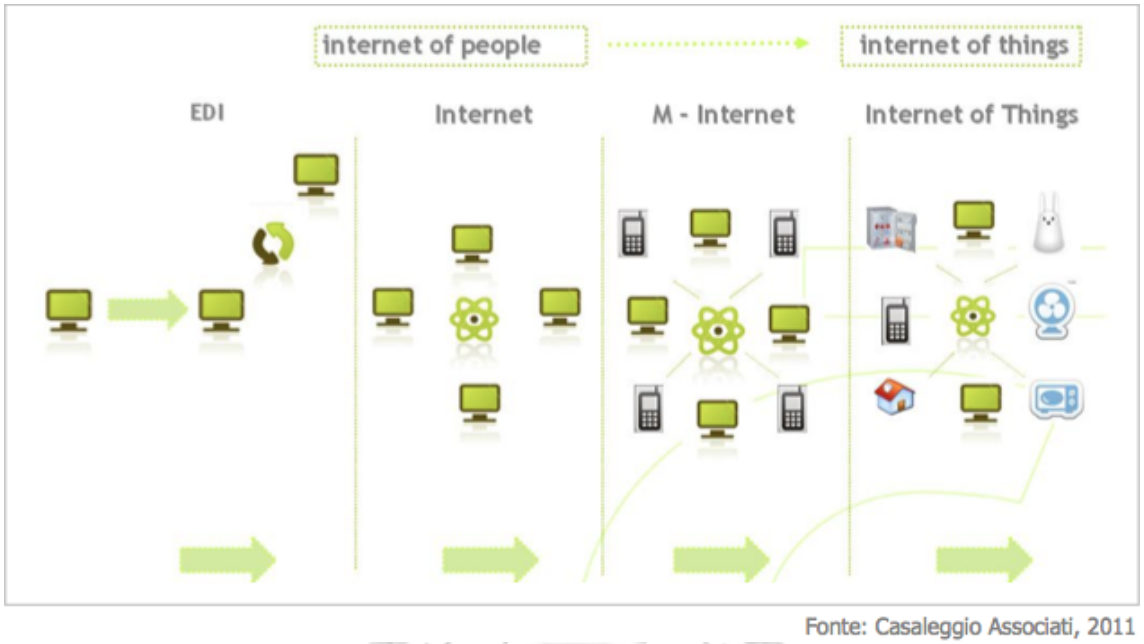


Figure 2.1: The Evolution from Internet of People to Internet of Things

1. Smart Home: At home, embedded sensors can sense human presence and could understand human activities to properly adjust air quality and temperature by commanding the air conditioner on or off to save energy without sacrificing human comfort.

2. Economical agriculture: In a vast farm land, remote sensors can detect and identify the outbreak of pests and initiate spreading pesticide at only the areas of outbreak.

3. Vehicle safety: Sensors on the car can help drivers understand its own driving conditions and monitor driver condition to help others to understand driver's condition to prevent traffic accidents.

In conclusion, connected embedded things could enhance people's lives tremendously in a way never seen before. This paradigm shift creates numerous opportunities and challenges for engineering [21].

2.1.4 Challenges

In the future, enormous amount of sensors, actuators, or things will be deployed. The cost of serving such amount of things will be a major concern. One big challenge is to reduce and minimize costs to deploy and maintain the system. Many domestic ubiquitous computing projects have failed because of high complexity in sensor network deployment [22].

Once the network of sensors have been deployed, it is almost always impossible to perform manual effort to repair such as battery replacement, case replacement. Therefore, another challenge is low power sensor design, or self-sustainable design eliminating the needs for battery over the lifetime of sensors. For example, if the sensors are deployed in mountain range where human contact is costly, the lifetime of sensors should last as long as the lifetime of the project.

After the data has been collected, the sensors will communicate the data collected with other parts of the network. Since the number of sensors have far exceed the number of humans, and this gap will only accelerate faster in the future [18], it is a challenge to serve the huge amount of data generated by sensors when the base station can only provide quality of service up to an arbitrary threshold. The ability to provide critical analytics to the ocean of data generated by IoT will be a key role in the new computing era.

2.2 Wireless Sensor Networks

2.2.1 What is Wireless Sensor Networks

Wireless sensor network (WSN), a M2M network, consists of a network of sensor nodes, where each is uniquely identifiable. Each sensor node is equipped with low-power,

low-cost, and failure-prone sensors or actuators. Sensor nodes collaborate to collect, process and disseminate environmental information[23].

Sensor network could be homogeneous, meaning all nodes are identical with same sensors, actuators and hardware setup. Sensor networks could also be heterogeneous where nodes have different sensors, actuators and hardware setup. Heterogeneous networks require higher level management and organization resources. Wireless sensor nodes could communicate wirelessly through air by sending electronic signals. Wireless communications aren't stable, as it is highly influenced by environmental factors.

2.2.2 Benefits of Decentralized Architecture

In general, centralized architecture is easy to implement but if the central node failed, the whole system would went down. A decentralized architecture might be more expensive to design, implement and more delicate to manage, but it is easier to scale and become more robust. In a decentralized architecture, each node capture a localized measurement. As the number of sensor node increases, the system could provide much greater accuracy and finer resolution in the information gathered compared to what a single sensor would obtain over a period of time.

In addition, new sensor nodes could be added to the network dynamically, and the system could continue to provide services. If a sensor node failed, nearby nodes could substitute for it and keep capturing the local information. Redundancy, or failover mechanism, is desirable as it makes the system more fault tolerant.

2.2.3 Challenges

Most of the challenges in WSN are similar to the challenges in building IoT system. However, in addition to the challenges above, it is difficult to develop a complex distributed applications with low end hardware and low end programming language. Most programming environments have been designed to work for a monolithic system that executes one instruction at a time in contrast to distributed systems with multiple instances of the program running asynchronously and each is prone to failure.

2.2.3.1 Service-Oriented Architecture for Wireless Sensor Network

Service-oriented architecture (SOA) is an established approach which eases the development of complex distributed applications. Using the concept, sensor network applications are composed of successively of independent software building blocks (services) which are situated on the sensor nodes in order to perform predetermined tasks. [14, 24] SOA provides an easy-to-use interaction paradigm and decouple the application from the low-level technologies.

2.3 WuKong: The intelligent middleware for M2M applications

2.3.1 Goal

Deployment and development for M2M applications are in its infancy today, as many applications are still single purpose in homogeneous networks with specific network protocols. The hardware has a fixed range of sensors, and the applications cannot be easily

ported to other platforms.

The existing middleware support that decouple high-level application design abstractions and low-level hardware has not been successful.

In Intel-NTU Center Special Interest Group for Context Analysis and Management (SIGCAM), we have been collaborating on a project, called WuKong, aiming to develop an intelligent middleware for developing, and deploying machine-to-machine (M2M) applications with ease. The main goal of this project is to support intelligent mapping from a high-level logical component flow (FBP) to self-discovered devices with selected network connections a target environment[11].

2.3.2 Flow Base Programming

M2M applications are by definition distributed where the application requirements involve a network of nodes collaborating for some common goals. M2M applications are typically defined by its flow of information between components, as opposed to more traditional applications that focus more on local information processing.

Flow Base Programming (FBP) is best suited for describing M2M applications as it allows the developers for the applications to focus more on the abstraction meaning of the components instead letting the unimportant details such as the hardware to stick right in the face. The result application will contain all necessary information for the framework to construct low-level details to implement the flow.

Applications are designed and constructed on FBP canvas by dragging a set of abstract components from the library as illustrated in figure 2.2. Each component is illustrated by a green block, each block has a set of properties, each with different access modes, such as *readonly*, *writeonly*, *readwrite*. Properties on the left of the green blocks are properties

that could be written, and properties on the right are readable. Components are connected by links, which is drawn by linking two properties in different components.

Some components represent physical hardware such as sensors, or actuators while some other components represent virtual processes such as mathematical computations, comparisons, etc. However, the final physical implementations of the components are only made during application deployment by the Master but not during FBP construction.

Components expose their interface through properties. A link can only made with properties with matching data type. The FBP application in figure 2.2 illustrates a simple scenario where the light actuator will turn on the light if light level drops below some value. The Numeric Controller component will be assigned to a user input device used by users to set its desire light threshold, which its output is sent to Threshold component. The light value is sensed from Light Sensor component and sent to Threshold. If the light value sensed is below the threshold value, Threshold will output a boolean to set the on off property of Light Actuator to turn the device, which will be determined during deployment, that it is represented by on or off.

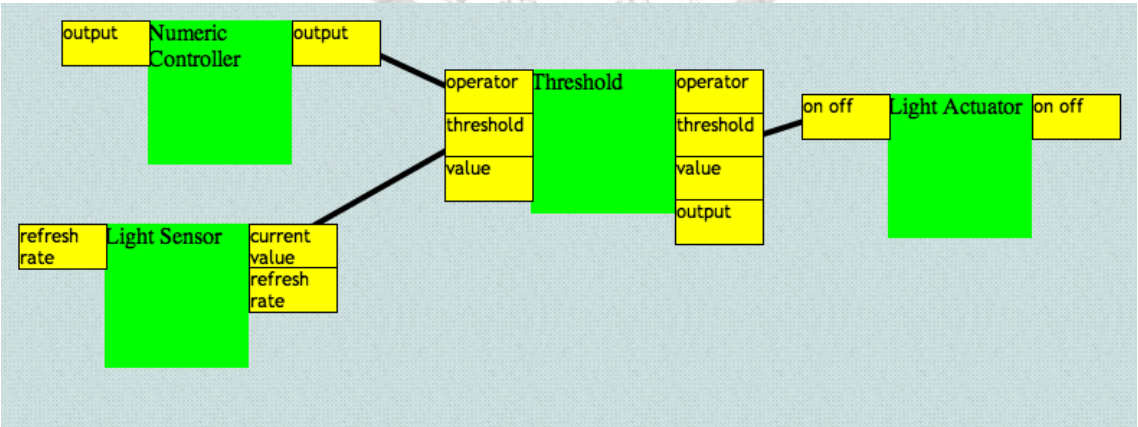


Figure 2.2: A FBP application

2.3.3 Sensor Profile Framework

While FBP defines the logical view of an application, Sensor Profile Framework allows tracking, identification of physical resources within the sensor network. There are a range of sensors which provide similar functionality with different level of quality. Sensor Profile Framework could model the sensor capability to enable handling heterogeneous sensors and provide a common abstraction for the logical view.

There are two main concepts in Sensor Profile Framework, WuClasses and WuObjects. WuClasss model components by exposing a number of properties describing, and allow access to, a specific resource represented by the class. For example, from drawing from the example in Figure 2.2, the on off property of Light Actuator component is boolean writeonly. WuClass also implements an update function to describe a component's behavior. For example, Threshold has four properties: operator, threshold, value, output. The output value is determined from the previous 3 properties that it returns true when the value is lower or higher than the threshold which depends on the value of the operator, and it returns false otherwise.

WuObjects are the main unit of processing that are hosted on the nodes. Each WuObject is an instance of a WuClass. Both concepts allow the framework to achieve 4 responsibilities:

1. Allow the Master to discover the current status of a node with the list of WuClasses and WuObjects it has.

2. Create new WuObject instances on a node to start receiving data and doing local data processing.

3. Trigger executions in WuObjects, either periodically or as a result of changing in-

puts.

4. Propagate changes of properties between linked properties in different components, which may be hosted locally or remotely.

2.3.3.1 Property Propagation

Sensor Profile Framework is in charge of communication between WuObjects as well, which are not necessarily on the same nodes. It monitors the changes in properties and propagates the changes to the connected WuObjects. For example, if a Temperature WuObject is connected to a Threshold WuObject, the changes in Temperature current value property will trigger propagation from the framework to propagate the new value in current value to the Threshold WuObject connected property, and since Threshold WuObject could be on a different node, the framework will take care of this by initiating a wireless connection between the nodes to send the data over. Once a new value has been set, Threshold WuObject will also trigger its update() function to recompute its output properties which in turn would cause another chain of propagation to the linked WuObjects.

2.3.4 Compilation and Mapping

Figure 2.3 shows the overview of WuKong build flow. The left part represents the build process for NanoKong VM which will be installed on the sensor nodes as part of the WuKong framework. The top part represents the build process for generating component libraries and Virtual WuClass library which will be used in other parts of the process. The right part illustrates the build process for FBP applications from being drawn in the IDE to Java bytecode that will be transmitted to the nodes.

The FBP program from the IDE will be exported as XML to the Master, the Master

will then take this XML and passed to Mapper to generate a Java program that will be executed on the nodes. Finally, the compiled code is then wirelessly uploaded to the nodes in the network.

The Java code consists of many parts from different phases of the mapping process. First, the Java code contains information about links between components that were taken from the FBP XML passed in earlier from the IDE. A link contains the source component id, destination component id. The library code for components corresponding to the component ids are stored in the node if it is written in native language, or uploaded as part of the Java bytecode if it is written in Java language. Second, it contains information about the mapping from application component ids to actual node identifications and positions. The purpose of a mapping which separates from the actual link makes it easier to substitute the actual host of the WuObject later during reconfiguration from the Master. This mapping is created by the Master during discovery phase that probe the network for node's capabilities in terms of available WuClasses, then mapper will decide the final candidates that will be hosting for a component. If no native version of a component is found on the nodes, mapper will substitute with a Java version of it.

2.3.5 System Progression Framework

There are a few popular wireless communication protocols in M2M applications: ZigBee, ZWave. It is expected that in the future more diverse wireless nodes equipped with radios that support protocols such as low-power Bluetooth and WiFi that all have one or more powerful gateway to connect to the outside world. In WuKong system, one of the gateways will take on the role of higher management decision maker called *Master* to making the decisions for deployment and producing the configuration of WSNs.

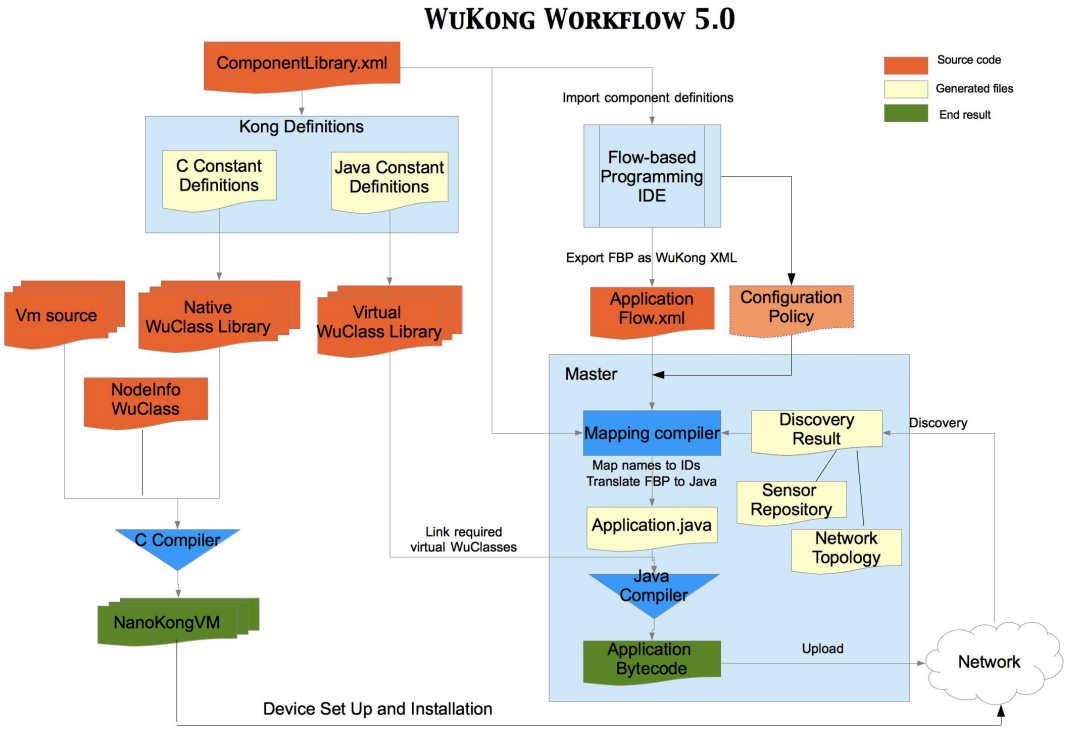


Figure 2.3: WuKong application build flow

Chapter 3

SYSTEM DESIGN

The design for the fault tolerance primitive is guided by the goals and challenges outlined in chapter 1: We want fault tolerance policy to be expressive, component specific, yet decoupled from physical hardware specifications so it will work with a range of network configurations. Furthermore, we want a reconfigurable redundancy architecture for service-oriented heterogeneous WSNs so components would be resilient to partial failures and degrade gracefully as indicated in the policy while being efficient and simple in terms of engineering design.

3.1 User Preference for Fault Tolerance

Figure 3.1: An example of categories a user policy could impose on a component

User policy guides the decisions and achieves certain outcomes for the system. Specifically, user policy controls how the system could behave in high level concept. When

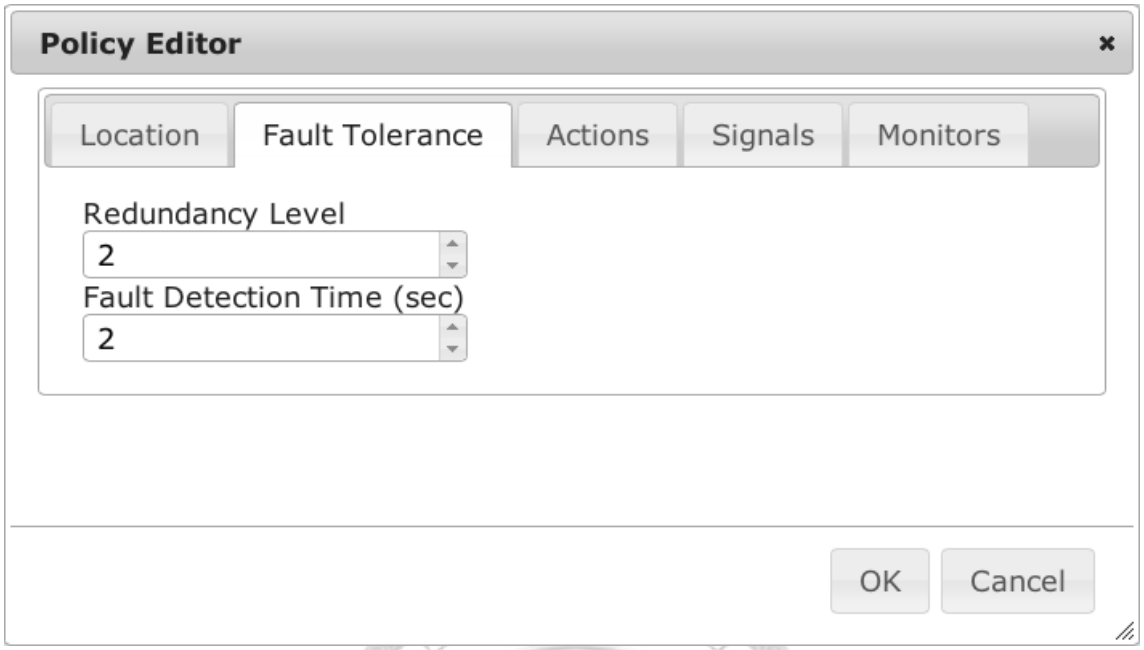


Figure 3.2: An example of fault tolerance policy

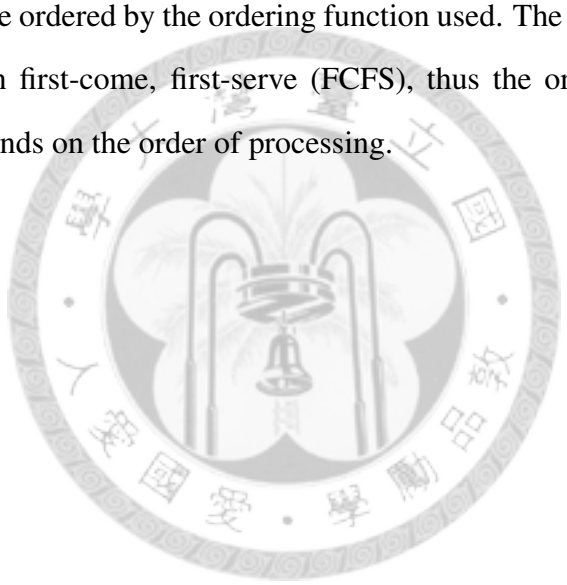
users starts up WuKong user interface, they will be confronted with the application where users could specify policy for each application component on, for instance, specific location the application component should be mapped to, or number of redundant devices the application component would have as illustrated in figure 3.1.

Users are also capable of specifying fault tolerance requirements via the user policy for fault tolerance as shown in figure 3.2 Redundancy level indicates the number of devices that will be able to take over when the service failed. Fault Detection Time represents the time the system should take approprimately to detect an failure in seconds.

3.2 Deploying Application with Fault Tolerance

In WuKong, deployment consists of discovering available resources, converting application from high-level abstractions to low-level machine instructions, determining the

system parameters based on user policy, and then finally deploying the application to the network. The process was briefly described in section 2, however, the thesis has made some significant changes to the process to support strip, an redundancy abstraction used in failure recovery and reconfiguration algorithms. First, we add a new process in mapper to take network topology results from discovered sensor network, so we could arrange heartbeat groups for heartbeat protocol. Next we take the existing mapping process, and convert it to outputting strips. So instead of one-to-one mapping from components to devices, now each component maps to a strip, which could be seen as a list of devices supporting such service ordered by the ordering function used. The default ordering function uses first-fit as in first-come, first-serve (FCFS), thus the ordering of the strip is randomized, as it depends on the order of processing.



A first-fit algorithm for mapping is shown in algorithm 3.2.1. Algorithm 3.2.1 takes two arguments, C and N. C is the set of components; each component has a *strip* attribute representing a list of nodes assigned to it. N is the set of node infos; each node info has a *services* attribute representing a set of components on the node. This algorithm outputs a mapping of S, a set of components with mapping to a list of nodes. Once the mapping is produced, application will be deployed and each device would be reprogrammed wirelessly to create corresponding WuObjects to host the services.

Algorithm 3.2.1: FIRSTFITCOMPONENTTOSTRIPMAPPING(C, N)

$S \leftarrow \emptyset$

for each $c \in C, n \in N$

do
- $s \leftarrow c$
- **if** $s \in n.services$
  - **then** $s.strip << n$
- $S \leftarrow S \cup \{s\}$

return (S)

3.3 Strip

Representing a component in WuKong application, each Strip contains a list of node on which the WuObjects representing the component are hosted. As seen from the figure 3.3, each node represented by a big white block contains a copy of strips for components that are specified to have redundancy in fault tolerance policy. Nodes holding a duplicated WuObject are members of the strip. The membership of a strip is called the view. Only the WuObject hosting at the head of the strip will be active, while the rest are backups. As seen from figure 3.3, node 1 has a active WuObject for component A as shown highlighted in yellow, but duplicated WuObjects A in node 2 and 3 are inactive in grey. In Strips, when one member failed, the next one will take over. For instance, if a strip is constructed like this $\rightarrow 1-2-3-4-5$, when 3 failed 4 will take over the place of 3, and the new chain will look like this: $\rightarrow 1-2-4-5$. Now if 1 failed, 2 will take over which would result in $\rightarrow 2-4-5$, since node 2 is now at the head of the strip, its WuObject will be active. Typically, there would be multiple components deployed to the

network and each node could carry multiple WuObjects, therefore many of these strips in the network would crisscross with one and others. It is not unlikely to see a node carrying active and inactive WuObjects at once.

3.4 Reconfigurable Redundancy Architecture

Once the application is deployed, each target device would host one or many WuObjects where each represents a service for an application component. However, failure would occur, so the network has to be resilient to failures: network has to be able to detect failure and recover autonomously. In the following sections, we will be describing the subsystems used to support failure detection and recovery.

3.4.1 Decentralized Failure Detection

We want failure detection to reach the entire network so no failures would be left undetected. In order to avoid single point of failures and to scale to larger networks, the protocol will have to be decentralized, thus failure of one system component would not bring the whole failure detection system down and would have lower detection latency.

Decentralized failure detection enables high-availability in distributed systems where partial failures is rather common. We utilize heartbeats to detect failures; a common technique widely used to detect failures in high-availability distributed systems. Heartbeats are messages sent periodically until it's unable to send messages anymore. Each node is therefore suspected dead when others stopped receiving messages from it after an extended period of time. Nodes were assumed to fail by stopping and will never come back.

There are abundant literature on designing heartbeat protocols to ensure high-availability for distributed systems. Our work employed a heartbeat protocol arranged in such as way where each node sends a heartbeat to the previous node and the last node sends back to the first node forming a daisy chain as represented by the black arrows in figure 3.3.

To prevent tight coupling and redundancy in heartbeats, strips are separated from heartbeats so the order of the strip does not affect the ordering of heartbeats and vice versa. The heartbeat protocol is a support layer below strips, the layers above will take advantage of the given information from the layer below to recover the system.

3.4.2 Failure Recovery

When a failure is detected, there are two tasks that the system would have to do to recover from failures. First, it has to make sure all members that carries the strips in the failure nodes will have consistent view of the strips. Second, it would need to propagate the changes to reconfigure other parts of the system that depend on the locations of the heads of the affected strips in order to function. The details of each task is described in the following sections.

3.4.2.1 Consistent view of strips

Consistent view of strips is required to pick a replacement node in the event of failure. Without consistent view, nodes will not be able to know if the component has been replaced. For example, in figure 3.4, when node 2 failed, node 1 will detect this, but without algorithms to maintain consistency, node 3 would not be certain if node 1 or 2 are still alive, thus it might take actions that could compromise the network.

But detector might not know which strips the monitoring node is a member of, so

every node will have a copy of its monitoring node's strips view as shown in figure 3.3 below the "Monitored Strips" section where node 1 has knowledge of the strips views of its monitoring node 2.

The detector of the failure would initiate the recovery algorithms. Since the detector will be responsible for recovering for the failed node, every node needs to have membership knowledge of the strips from the nodes it is monitoring. For example, if node A is monitoring node B, A would know the members of all strips in node B in addition to its local strips. Strips only specifies the order of recovery, it is not correlated with the network structure for the fault detection, in other words, a strip with A and B doesn't mean B is monitoring A, as B could be monitored by C which depends on the structure of heartbeat protocol layer. In the initial algorithm, the detector node will prepare a update message to inform all members of the strips with which the failed node is associated with. Assuming that every node that monitors other node will have knowledge of the strips that it contains and the members that the strips pertain. The node would send out a marker message first to confirm the nodes which are still functioning, and once all acknowledges have been received, it will proceed to send the update message to update their local knowledge of the strips to reach a consensus. The ordering of the messages wouldn't matter since the end state of any failure sequence for any strip would be the same. For example, given a strip of three members $\rightarrow 1-2-3$, if the updated failure sequence is given in any permutation by $[1,2]$ or $[2,1]$, the end results would be the same $\rightarrow 3$ since the remaining members from those two failure sequence is the same and the relative order of the members would stay the same. Therefore there is no need for extra communication overhead to maintain ordering to guarantee level of consistency between members since they will all come to the same conclusion given each receiver receive the same messages. The overhead are messages required to update each member's internal membership information.

3.4.2.2 Reconfiguration

After the consensus of the new view for all strips in the failure nodes has been reached, other devices in the network with connected components would also need to be updated. After finishing synchronizing views, the detector would initiate the reconfiguration algorithm. First, it would identify application components that are connected with the components carried by the failed device (linked in the FBP). Then it would issue the update message to each head of the strips of the connected components to reconfigure the new heads of the affected strips.

As shown in figure 3.5, node 1 with its updated view of the strips after the failure of node 2 would need to update the new head of component A, which is still node 1 in this case, with the new head of component B since component B is connected to component A; node 1 also needs to update the new head of component B with the new head of component A as well. Thus node 1 will send a reconfiguration message to node 1 and node 6 about the changes in both heads of component A and component B.

To keep the nodes updated after reconfiguration, as most services in WuKong applications does not store any past data, the devices with connected components whose WuObjects originally are sending data to the WuObjects on the failed devices would force a data push to bring the new heads up to date, and vice versa, so instead the devices with connected components whose WuObjects originally are receiving data from the failed devices would force a data pull.

The detector needs to update its knowledge of the ”Monitored” section after reconfiguration. Since it knows which node the monitoring node is monitoring, it would instruct the node to reroute heartbeats to itself, and then it would send a request to update its ”Monitored” section from the knowledge of the node such as local strips and the node

it is monitoring. As illustrated in figure 3.6, node 1 will get updated with the heartbeat information from node 3, which is monitored by node 2. Also in figure 3.7, node 1 would get updated with the strip information from node 3.

3.5 Complexity Analysis

The lower bound of message overhead for heartbeats arranged in a daisy chain loop in a network of size n is $\Omega(n)$, the upper bound is also $O(n)$, since there is only one message sent by a node at any given moment.

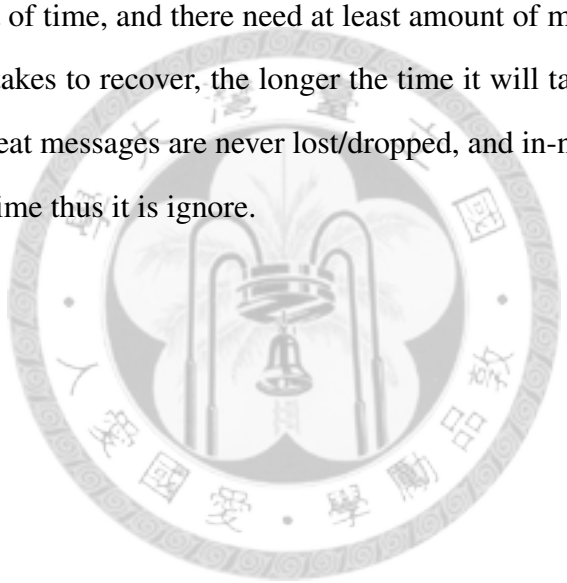
For an application with m components deployed to a network with n nodes where $n > 1$, the lower bound of memory overhead of the strips for a random member in the network is $\Omega(1)$ since a node must carry at least one strip and lowest member requirement of a strip is 2. The upper bound is $O(m*n)$ since the strips could span the whole network. But in reality, since the nodes typically have fixed memory and fixed capability for the duration of the lifetime in the network, the upper bounds cannot grow indefinitely. The memory overhead a strip could impose on a node is determined by node's capability and memory size.

The lower bound of message overhead by the reconfiguration protocol with one node failure in the same network is $\Omega(1)$, since if there is only one component with 2 members on the failed node, the detector would only need to send 1 messages to the remaining strip member of the failing node, and none if the component is not connected to any other components, and finally 1 more message to get the information from the nodes monitored by the failed node. The upper bound is $O(n + m)$ since if the failed node carried m components where each strip contains n members, the detector would have to send (n-2) messages to all functioning members of the m strips carried excluding itself plus the

messages to all other strip heads connected to the failing node.

The time to recover highly depends on the messages sent for the reconfiguration and the detection time. The lower bound is $\Omega(1)$ with the same assumption with constant component and members, but the upper bound is $O(b + n + m)$ where b is the heartbeat period.

The cost for time for recover is the time itself. Time is the overhead. Thus the longer time it take to recover the higher the overhead. By setting a shorter heartbeat period, it would take a shorter time to recover, thus a lower overhead. I assume every message takes at least a fixed amount of time, and there need at least amount of messages to recover, so the more messages it takes to recover, the longer the time it will take to recover. Here it is assumed that heartbeat messages are never lost/dropped, and in-node computation take negligible amount of time thus it is ignore.



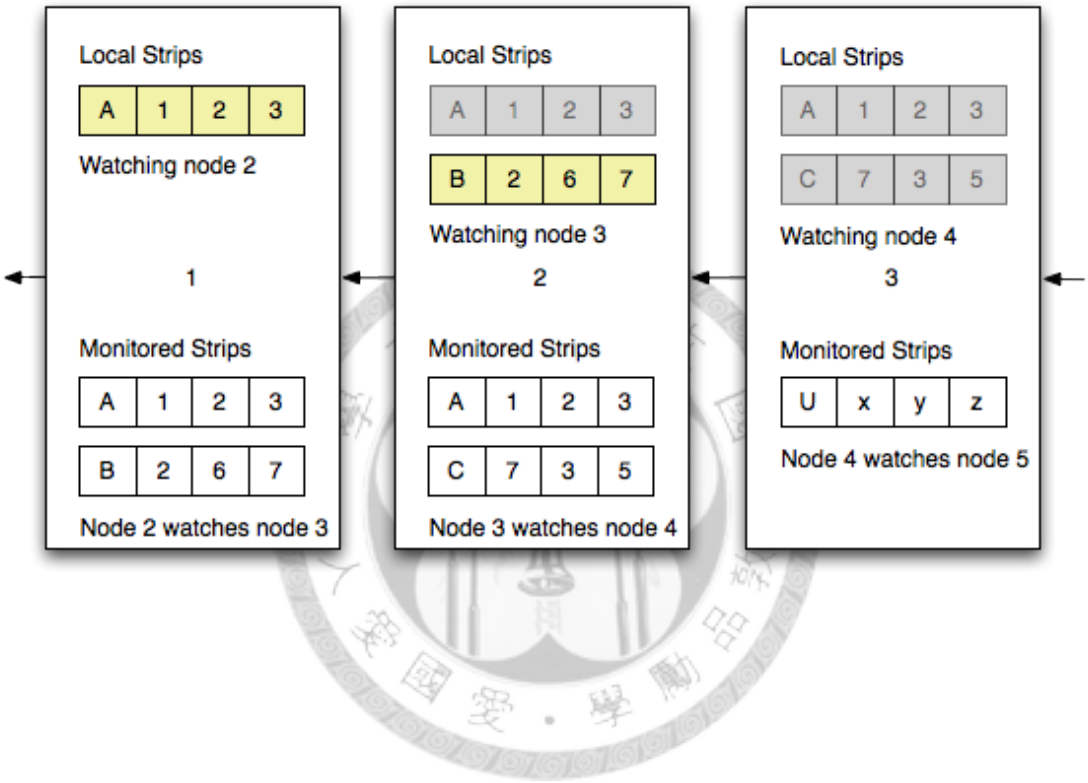


Figure 3.3: An example network with several strips and heartbeat protocol forming a daisy loop chaining node 3 to 1

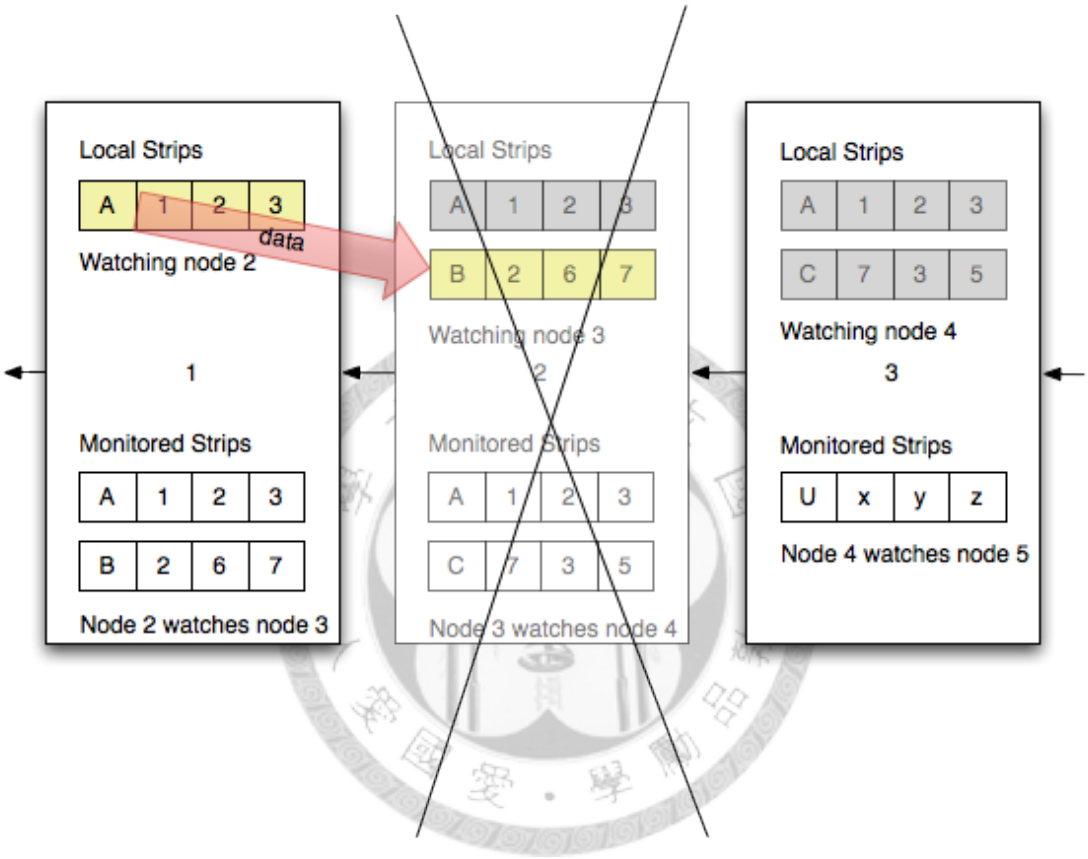


Figure 3.4: A failure occurred at node 2 in the network

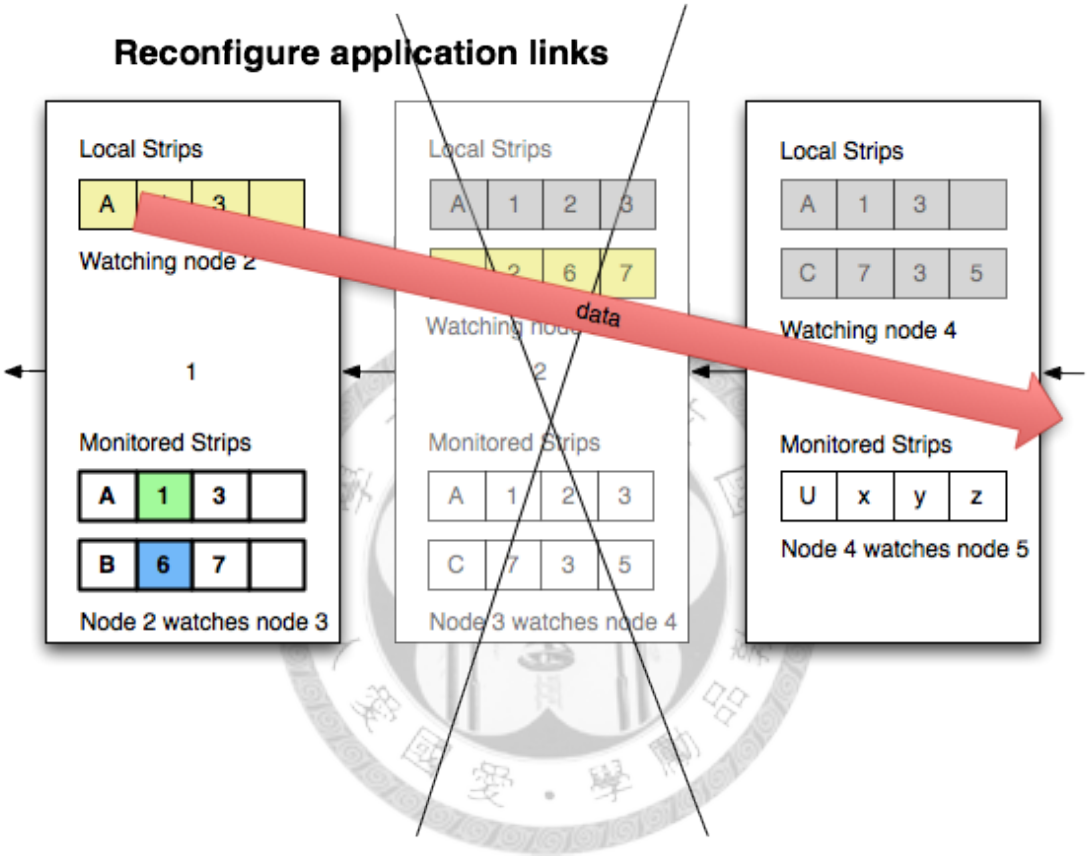


Figure 3.5: Reconfigure application links

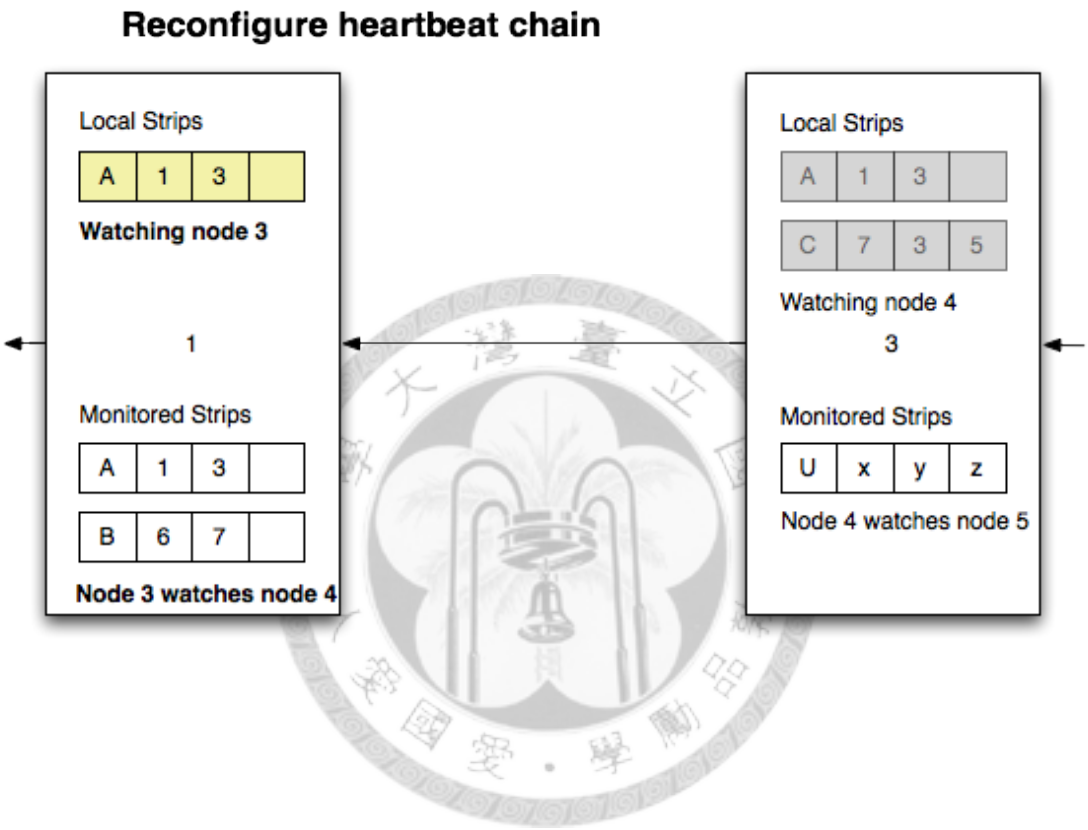


Figure 3.6: Reconfigure heartbeat protocols

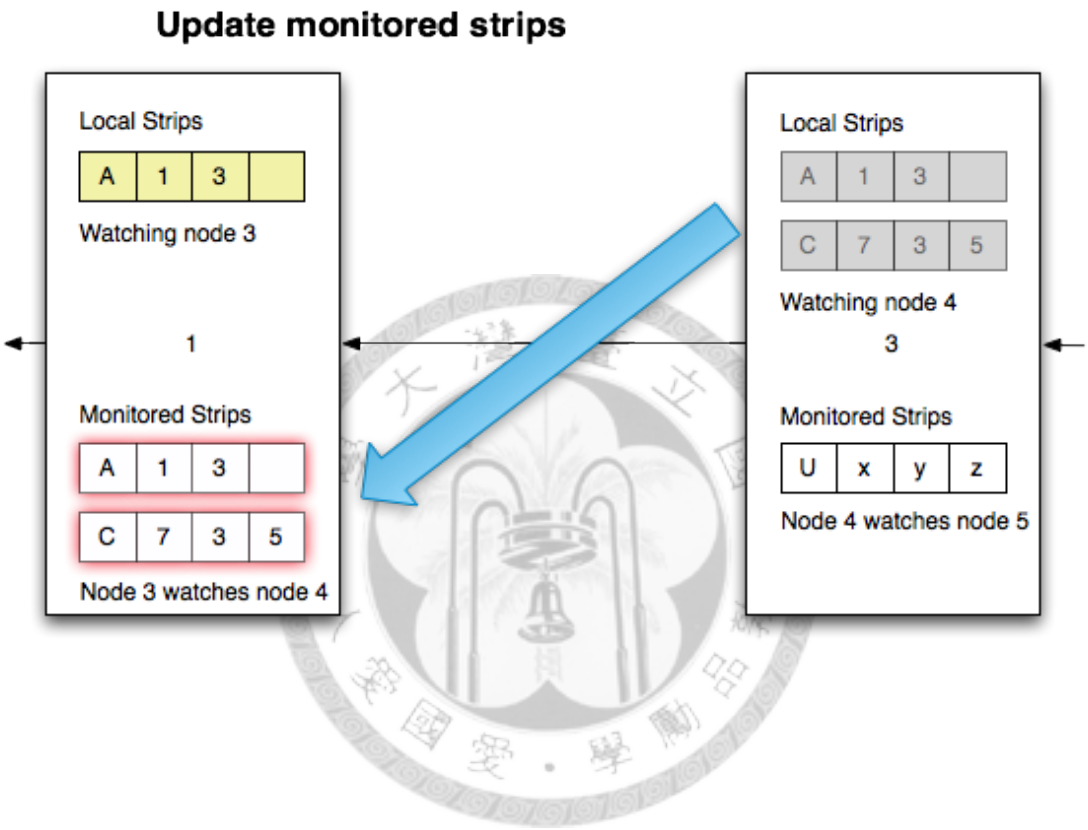


Figure 3.7: Update monitored information

Chapter 4

EVALUATION & RESULTS

In order to evaluate the performance of the system, we have introduced some metrics to test how well it would perform, and whether after node failures requirements could still be met.

1. Whether failover kickoff successfully after failure
2. Memory overhead for Strips
3. Message overhead for failure recovery, including reconfiguration
4. Time to detect and recover from failure

We measured system performance live by collecting data from sensor nodes while running. Sensor nodes were programmed to send out their tracking data to a central data sink at appropriate times such as after node initialization or when the failure was resolved. The application, fault tolerance policy, network topology were described in the following sections.

4.1 Application

Application as shown in figure 2.2 were deployed. It consisted of four components:

1. Numeric Controller component represents a user input device which outputs a number from 0 to 255.

2. Threshold component represents a conditional function which takes two inputs, Threshold and Value, and, depending on the Operator attribute, return true if the Operator was set to GT (Greater Than) and the Value was higher than the Threshold value.

3. Light Actuator component represents a light source or a relay intercepting the power source for a light source. It has a property OnOff which turns on the light if it was set to true, otherwise the light will be turned off.

4. Light Sensor represents a photodetector sensor which detects the level of light intensity.

4.2 Policy

Component fault tolerance policy for the application was set with the following parameters:

Numeric Controller component

Redundancy Level: 1

Fault Detection Time: 2 sec

Light Sensor component

Redundancy Level: 2

Fault Detection Time: 2 sec

Threshold component

Redundancy Level: 1

Fault Detection Time: 2 sec

Light Actuator component

Redundancy Level: 9

Fault Detection Time: 2 sec

In order to reduce the probability of getting false positives caused by environmental factors such as poor wireless communication quality, timeouts for heartbeat were set accordingly. We set it to right around double of the heartbeat period we set for the network. Thus, assuming the worst time to detect failure takes the full length of fault detection time, the heartbeat period was therefore 1 sec, which was one half of the fault detection time.

4.3 Heartbeat Protocol Arrangement

The devices were deployed in proximity to each other in a room, forming a fully connected network. The heartbeat protocol arrangement was arranged in a loop so every node was sending heartbeat to previous node except the first node, which sends to the last. For example, node 1 receives heartbeat message from node 2, node 2 receives heartbeat message from node 3, etc. Figure 4.1 illustrates the arrangement for this experiment.

4.4 Hardware Platform

All boards were equipped with an Atmel ATmega2560-16AU 8-bit microcontroller with 4K of EEPROM and 64k of flash. The figure 4.2 shows the board used in the experiments. The hardware design was based upon Arduino hardware referenced design; in addition, every board has wires for mounting multiple wireless protocol adapters supporting protocol standards such as ZWave [25]. ZWave is a wireless communication protocol designed for home automation, specifically to remotely controlled applications in residential and light commercial environments.

In the following experiment, every board was only equipped with a ZWave adapter and only communicating wirelessly through the ZWave adapter. In addition, every board was installed with "NanoKong" [26], which includes NanoVM [27] and WuKong Profile Framework that supports all the basic WuKong framework protocols including the new additions from the work in the previous chapter. A PC with wireless access was dedicated for hosting the WuKong Master software which was responsible for managing WuKong applications for the whole system and serves as a mean to present an interface to the users.

4.5 Experimental Setup

The complete system setup is shown in figure 4.3. Ten boards were used in the experiments. Two of them were equipped with a Arduino brick light sensor [28] that returns light intensity value from 0 to 1023. The rest were equipped with a onboard LED as a simple light source. Boards with light sensor were assigned a light sensor component. Boards with light actuator were assigned a light actuator component. One of them were assigned a Numeric Controller component, and Threshold component for the deployed

Table 4.1: Node setup

Node Id	Equipped Resources
1	Light Actuator
2	Numeric Controller, Threshold, Light Sensor
3	Light Actuator
4	Light Actuator
5	Light Actuator, Light Sensor
6	Light Actuator
7	Light Actuator
8	Light Actuator
9	Light Actuator
10	Light Actuator

application. Every WuDevice would be able to talk to each other directly through ZWave wireless adapter. The deployment formed a fully connected network. We simulated a node failure by removing power supply from a WuDevice. The detail of each node setup was shown in table 4.1.

4.6 Mapping Results

The result of the mapping and the strips were shown in table 4.2. Each row represented each component in the application, where strips were ordered from the left. Numeric Controller was mapped to only node 1; Light sensor was mapped to node 2 and 5, where 2 hold the active WuObjects when deployed; Light actuator was mapped to 9 nodes, and node 1 hold the active WuObject when deployed; Threshold was mapped to node 2. The result indicated that the only active nodes were 2, and 1 right after deployment.

Table 4.3 listed the memory overhead of each strip in the application. Each entry in the strip consumes 2 bytes, one byte for addressing and the other is for internal addressing for WuObjects. Strips for Light Actuator, for example, takes up 18 bytes because there

Table 4.2: Strips

Application Component	**Mapped nodes (strip)**
Numeric Controller	2
Light Sensor	2, 5
Light Actuator	1, 3, 4, 5, 6, 7, 8, 9, 10
Threshold	2

Table 4.3: Memory Overhead of Strips in Bytes

Application Component of Strip	**Memory size (bytes)**
Numeric Controller	2
Light Sensor	2
Light Actuator	18
Threshold	2

are 9 nodes assigned as members.

4.7 Results

The total memory overhead of a Light Actuator strip aggregated from its members at different member size were shown in figure 4.4. Each entry in strip takes two bytes as one byte is used by node address defined in ZWave protocol, and the other byte was used for our internal WuObject identification system to identify wuobject on a node. The growth of the overhead is quadratic, confirming the analysis in previous chapter that $O(m * n)$ when $m = n$.

Figure 4.5 showed message overhead used to recovery random Light Actuator strip member failure against different strip sizes. The message overhead contains incoming and outgoing messages, including retries, for the detector during the recovery process. The growth in the figure is linear, confirming the analysis in the previous chapter that $O(m + n)$.

The figure 4.7 illustrates the recovery time averaging over 5 deployments for each node failure in Strip for Light Actuator as first failure in the system. The first failure should on average takes the longest time compared to consequent failures, therefore, measuring the performance for each node failure as first failure would give us how the system would perform the worst overall. The results carried over different strip orders since the ordering was just a matter of permuting the results as shown in the results.

The recovery time for most nodes were averaging around 2500 milli-seconds; node 3 and node 6 were found out that their radios were a little defective (without antenna) after the experiments therefore it took longer to complete the recovery. It was clear that the results have shown were pretty consistent as there were only a constant number of nodes that needed to contact to recovery regardless of how many strips the node contained. The time it took was reasonable given the small network.

The results of detection time plus recovery time for different heartbeat periods are plotted in figure 4.6. Heartbeat period is the interval between heartbeat messages a node sent to another node for health monitoring. Detection time is the time it took to detect failure and initiate recovery algorithm by the detector. The results have shown a linear relationship between heartbeat period and total time to recovery from failure.

Throughout the experiments, we found out that none of the deployments failed, in other words, all failovers were successful and correct. The experimental results above have shown the feasibility of the system that it could reliably handle single failure at a time with great performance over a small network.

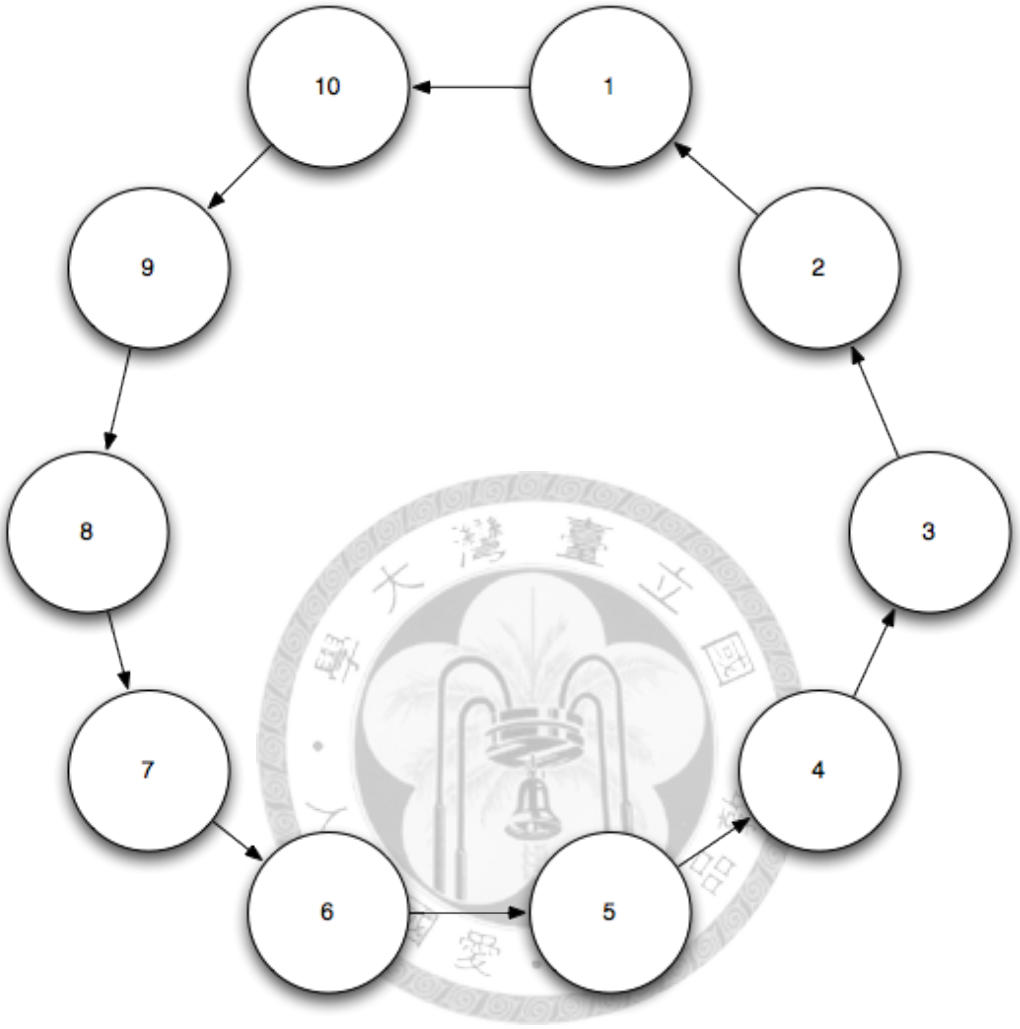


Figure 4.1: Heartbeat protocol arrangement where the arrows indicate the direction heartbeat messages get sent to

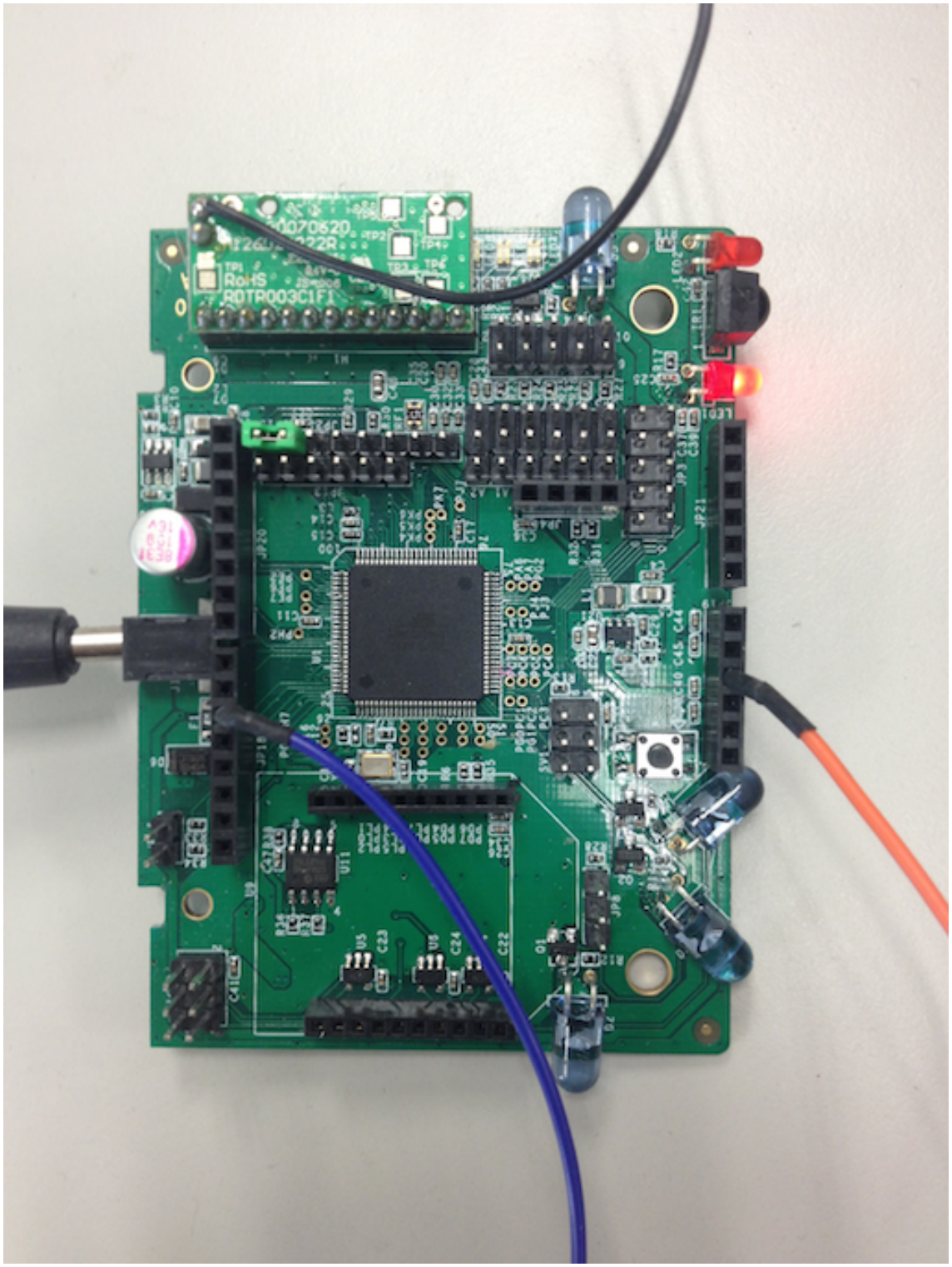


Figure 4.2: An WuDevice equipped with ZWave radio on the top, and running a ATmega2560 8-bit microcontroller in the middle

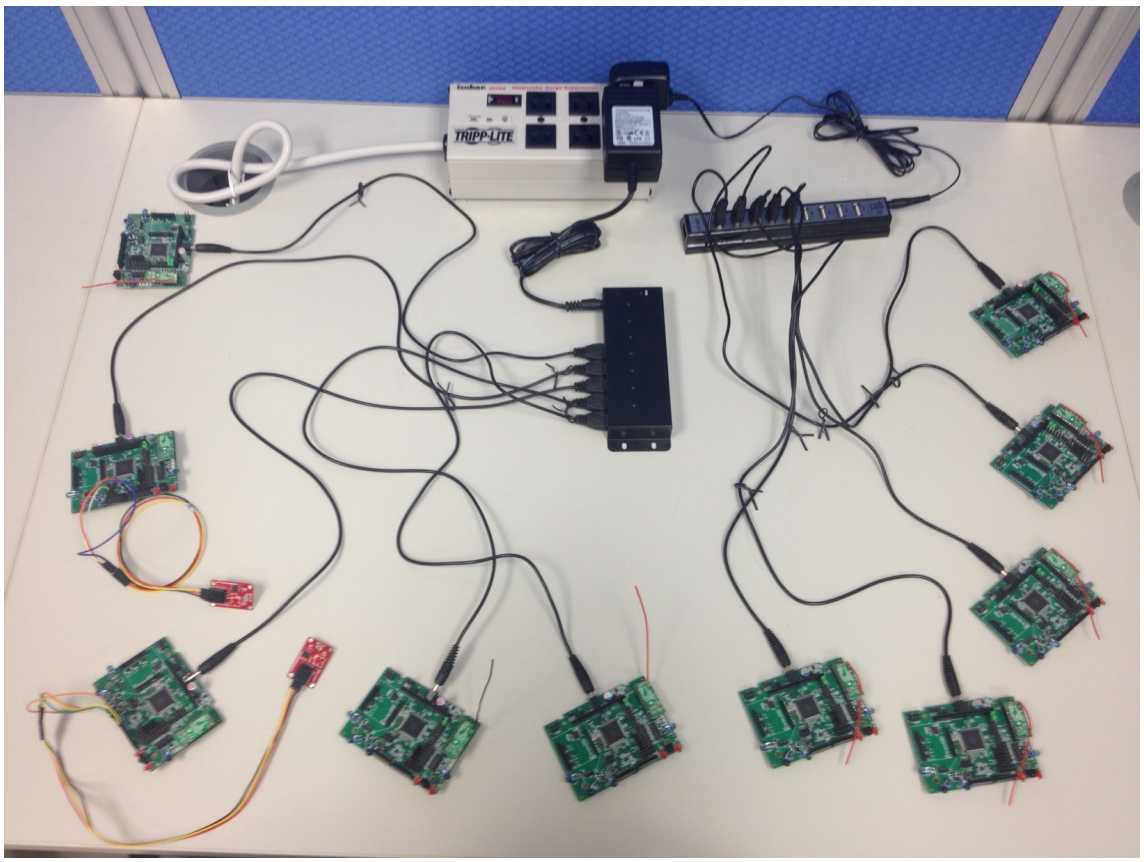


Figure 4.3: Whole system setup with ten nodes, two of them were equipped with light sensors

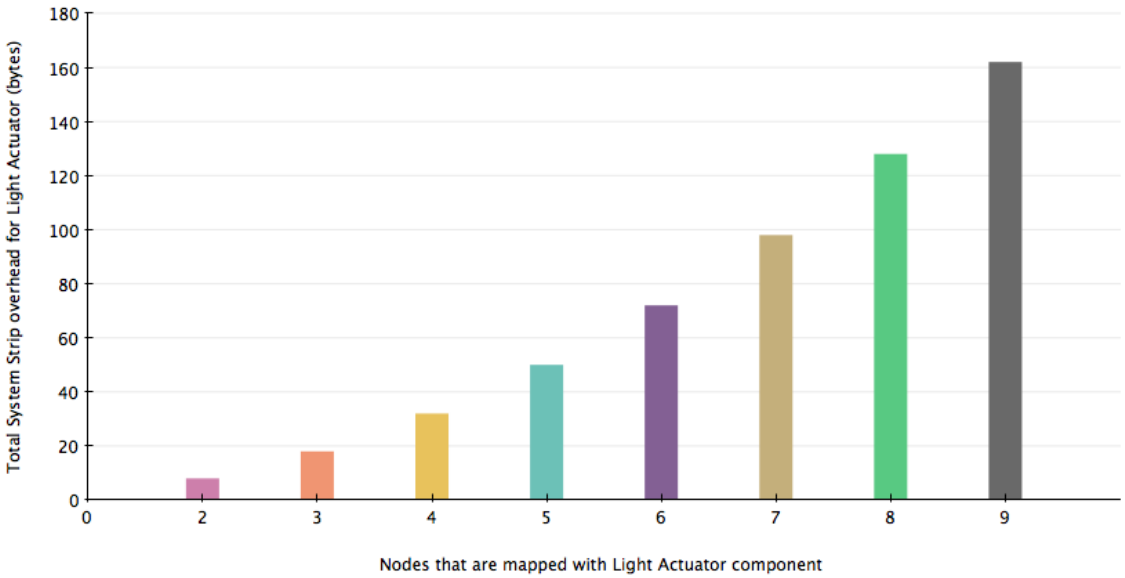


Figure 4.4: The total memory overhead of a Light Actuator strip aggregated among its members at different member size

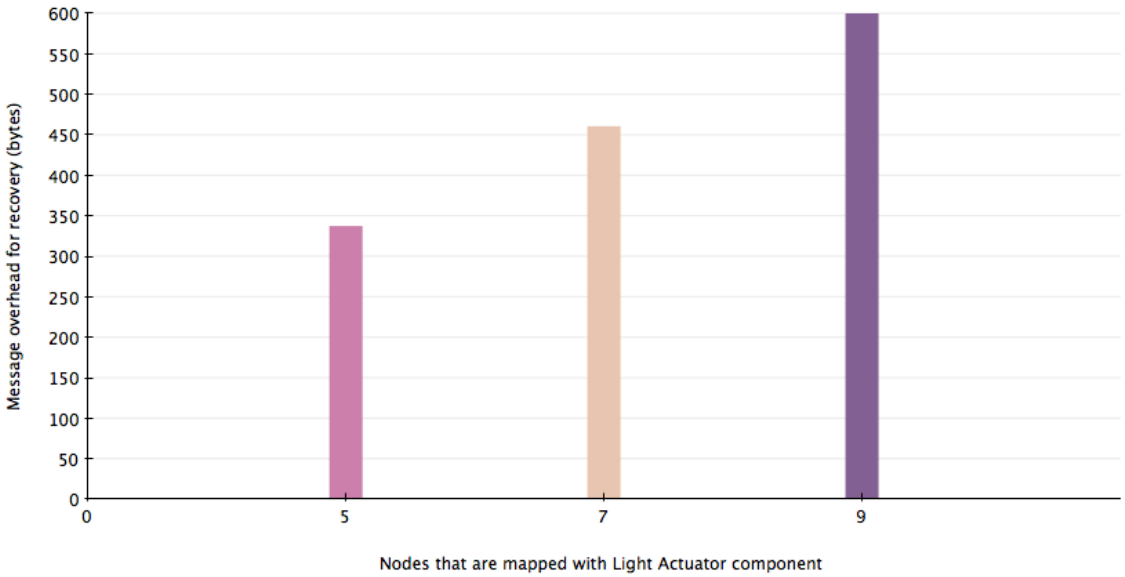


Figure 4.5: Message overhead for recovery of a random Light Actuator strip member failure against different strip sizes

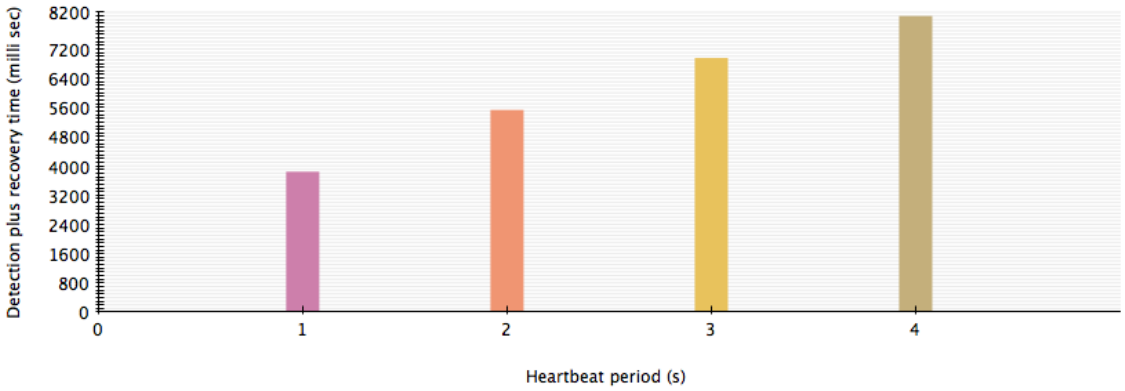


Figure 4.6: Detection plus recovery time in milli-seconds for heartbeat period in seconds

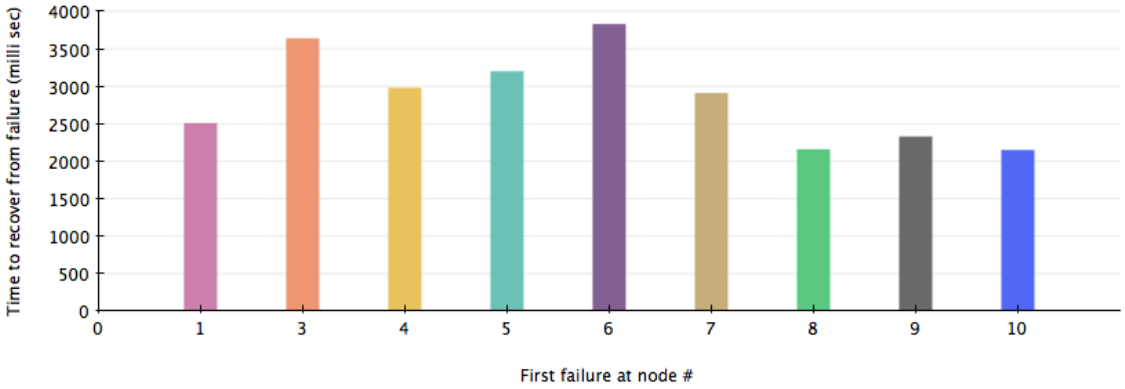


Figure 4.7: Average recovery time over 5 deployments for each node failure as the first failure

Chapter 5

CONCLUSION

5.1 Discussion

We have presented a fault tolerance primitive for WuKong that is able to configure itself based on user policy and requirements. We have described strip, a redundancy abstraction for service peers. We also described the reconfigurable redundancy architecture with distributed algorithms that synchronize strip view across network and perform network reconfiguration for the replaced services. It was shown in experiments that our approach is capable of failover in a small network.

The developed methods add new and useful solutions to build a fault tolerant primitive that could be reasoned easily and with a performance under expectation. Our system provides an extension to other solutions in terms of completeness and complexity. It serves as a quick and easy solution to provide practical failover for service-oriented applications.

5.2 Future Work

We have shown a design for a fault tolerance primitive for WuKong in which system could be configured based on user requirements, strips makes it really easy to describe a component system with redundancy for heterogeneous services and devices, and the

reconfigurable redundancy architecture ensure consistency after services recovery. Nevertheless, there is still room for improvements. This section will address some directions future research can take.

This study did not consider network partition. Network partition occurs when a network of nodes got partitioned into two subnetworks where none can detect each other for a period of time, but none of the nodes were dead. In this thesis we assumed fail-stop failures where nodes, once dead, will not come back; therefore, when network partition occurred, each part of the network would not be able to recognize each other and would cause conflicts and confusions. Our current heartbeat protocol is decentralized and easy to construct, but it is not shown to work under networks where messages are sent in multiple hops, since algorithms used to produce heartbeat protocol arrangement does not consider topology of the network. Heartbeat, by its nature, is sensitive to delay, so if a heartbeat message was not received within tolerance period, under the current implementation, the node would be suspected of failure and was assumed dead forever. However, if the node was still alive, since the revived nodes was treated as if it was dead, it would create an artificial network isolation, and that is a network partition. One of the possible directions is to create a more sophisticated failure model or heartbeat protocol that could handle network partition.

Current implmentation can only allow the detector to handle one failure at a time. It would be a desirable future research direction to investigate handling consecutive node failures. One possible way is by storing strips and heartbeat protocol data from nodes ahead, such that when consecutive nodes failure occurred the detector would be able to recover those services it backed up. However there is a tradeoff on the memory a node could store and the number of consecutive node failures a network could handle.

The optimization problem for application deployment is also an important element

in this system. This thesis didn't consider finding a optimal deployment for the level of redundancy specified in the user policy. The problem of deploying a specific distributed system onto a network structure typically consists of mapping the components of the system onto the hosts of the network. The mapping is subject to constraints. The constraints could be whether a node supports certain service to host certain components, and how much communication overhead would induce from the assignment to maintain consistency for the strips, and from the perspective of WuKong, some components need to separate from other components to achieve fault tolerance, and some needs to place together to function properly. Determining such an optimal deployment is a combinatorial optimization problem, and combinatorial optimization problems generally extremely challenging computationally. It is unlikely that a single approach will be effective on all problems or instances of the same problems. As we also want the system to come up with a solution within a time limit. So finding a good balance between the quality of a solution of the time it takes to come up with a good enough solution is critical.

Bibliography

[1] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli, "The hitchhiker's guide to successful wireless sensor network deployments," in *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08)*, vol. D, pp. 43–56, ACM Press, 2008.

[2] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson, "Analysis of wireless sensor networks for habitat monitoring," in *Wireless Sensor Networks* (C. S. Raghavendra, K. M. Sivalingam, and T. Znati, eds.), pp. 399–423, Kluwer Academic Publishers, 2004.

[3] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, "A line in the sand: A wireless sensor network for target detection, classification, and tracking," *Computer Networks*, vol. 46, pp. 605–634, Jan. 2004.

[4] J. Tateson, C. Roadknight, A. Gonzalez, T. Khan, S. Fitz, I. Henning, N. Boyd, C. Vincent, and I. Marshall, "Real World Issues in Deploying a Wireless Sensor Network for Oceanography," in *Unpublished paper presented at Workshop on Real-World Wireless Sensor Networks (REALWSN '05)*, 2005.

[5] P. Padhy, K. Martinez, A. Riddoch, H. L. . R. Ong, and J. K. Hart, "Glacial Environment Monitoring using Sensor Networks," in *Real-World Wireless Sensor Networks*, pp. 10–14, 2005.

[6] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline, "PIPENET: A Wireless Sensor Network for Pipeline Monitoring," in *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN '07)*, pp. 264–273, ACM, 2007.

[7] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in *Proceedings of the 3rd international conference on Embedded networked sensor systems (Sensys '05)*, pp. 51–63, ACM, Jan. 2005.

[8] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06)*, pp. 381–396, USENIX Association, Jan. 2006.

[9] X. Koutsoukos, M. Kushwaha, I. Amundson, S. Neema, and J. Sztipanovits, "OASiS: A Service-Oriented Architecture for Ambient-Aware Sensor Networks," in *Composition of Embedded Systems. Scientific and Industrial Issues*, pp. 125–149, Springer Berlin Heidelberg, 2007.

[10] D. Hughes, K. Thoelen, J. Maerien, N. Matthys, W. Horre, J. Del Cid, C. Huygens, S. Michiels, and W. Joosen, "LooCI: The Loosely-coupled Component Infrastructure," in *11th IEEE International Symposium on Network Computing and Applications (NCA 2012)*, pp. 236–243, IEEE, Aug. 2012.

[11] N. Reijers, K.-j. Lin, Y.-c. Wang, C.-s. Shih, and J. Y. Hsu, "Design of an Intelligent Middleware for Flexible Sensor Configuration in M2M Systems," in *Presented at 2nd International conference on sensor networks (SENSORNETS 2013)*, 2013.

[12] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications (WSNA '02)*, vol. 5, pp. 78–87, ACM, 2002.

[13] K. Piotrowski, P. Langendoerfer, and S. Peter, "tinyDSM: A highly reliable cooperative data storage for Wireless Sensor Networks," in *2009 International Symposium on Collaborative Technologies and Systems*, pp. 225–232, Ieee, 2009.

[14] J. Neumann, N. Hoeller, C. Reinke, and V. Linnemann, "Redundancy Infrastructure for Service-Oriented Wireless Sensor Networks," in *9th IEEE International Symposium on Network Computing and Applications (NCA 2010)*, pp. 269–274, IEEE Computer Society, July 2010.

[15] S. Shepard, *RFID: radio frequency identification*. McGraw-Hill New York, 2005.

[16] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[17] C. Associati, "The Evolution of Internet of Things," pp. 1–4, 2011.

[18] CISCO, "How the Internet of Things Will Change Everything." `http://youtu.be/mf7HxU0ZR_Q`.

[19] O. Malik, "Internet of things will have 24 billion devices by 2020," 2011.

[20] W. Randomly, "Supercomputers Vs Mobile Phones." http://www.walkingrandomly.com/?p=2684, 2010.

[21] Y.-K. Chen, "Challenges and opportunities of internet of things," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pp. 383–388, 2012.

[22] C. Beckmann, S. Consolvo, and A. Lamarca, "Some assembly required: Supporting end-user sensor installation in domestic ubiquitous computing environments," in *UbiComp 2004*, pp. 107–124, Springer-Verlag, 2004.

[23] A. Bharathidasan, V. An, and S. Ponduru, "Sensor Networks: An Overview," tech. rep., Department of Computer Science, University of California, Davis, 2002.

[24] R. Marin-Perianu, H. Scholten, and P. Havinga, "Prototyping Service Discovery and Usage in Wireless Sensor Networks," pp. 841–850, IEEE Computer Society, 2007.

[25] Z.-W. Alliance, "ZwaveStart." http://www.z-wave.com/modules/ZwaveStart/.

[26] P. Su, N. Reigers, and S. Zhou, "NanoKong." http://github.com/wukong-m2m/NanoKong, 2012.

[27] T. Harbaum, "The NanoVM: Java for the AVR." http://www.harbaum.org/till/nanovm/index.shtml, 2006.

[28] "Brick Light Sensor." http://arduino-info.wikispaces.com/Brick-LightSensor, 2013.