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

初稿

First draft

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 1 月

January, 2013

國立臺灣大學
資訊工程學系

碩士論文

初稿

蘇適撰

102
1

國立臺灣大學碩士學位論文
口試委員會審定書

初稿

First draft

本論文係蘇適君 (R99922157) 在國立臺灣大學資訊工程學研究所完成之碩士學位論文，於民國 102 年 1 月 21 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：

____

____

____

____

____

所 主 任 ______________________________

Acknowledgments

I’m glad to thank . . .

致謝

感謝…

中文摘要

Abstract

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of the thesis. First, we describe why WSN deployment and maintenance are still difficult, and how an intelligent middleware which employs a new programming model which separates design abstractions between high-level application design and low-level hardware constructs could be addressing the challenge. Next, we describe the needs for fault tolerance and why it is hard on such middleware. Lastly, we describe our proposed solution to this problem.

1.1 Motivation

1.1.1 Wireless Sensor Network Deployment is Hard

Wireless sensor networks are areas filled with network of tiny, resource limited sensors communicating wirelessly. Each sensor is capable of sensing the enviornment in its proximity. Wireless sensor networks are employed in a variety of applications ranging from home automation to millitary.

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. Deployment is a labor-intensize 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[3][12][2][17][11][14][18][19].

Testbed in laboratory environment can still not model the full extents of the influences a real world enviroment could do. Deployment still a big problem in wireless sensor network applications.

1.1.2 Maintaining WSN deployment

The possibilities of sensor failures is a fundamental characteristic of distributed applications. So it is critical for distributed applications to have the ability to detect and recover from failures automatically, since not only the sensor deployments are usually huge in scale, the deployment are to run unattended for a long period of time.

There have been abundance of research in fault tolerance for wireless sensor network deployments proposing low-level programming abstractions or framework to implement redundancy or replication.[20][10]

1.1.3 Component based middleware for distributed applications

However, the increase in number, size and complexity in WSN applications makes high-level programming an essential needs for development in WSN platforms.

This is supported by several reasons. Firstly, the diversity of hardware and software for WSN platform is as diverse as the programming models for such platforms[15]. Secondly, existing programming models usually sacrifice resource usage with efficiency, which is not suitable for tiny sensors in sensor networks. Thirdly, existing programming models still forces developers to learn low-level languages, which imposes an extra burden to developers, and it goes to show when the reusability for those programming models are low in existing applications.

Software componentization, or lightweight component models, has been recognized to tackle the concerns above. It brings several advantages over past approaches with separate of concern, module reusability, de-coupling, late binding.

The primary advantages of this approach is reconfigability, adaptability in applications like never before, since the high-level components and low-level constructs are loosely decoupled and interpretted by the middleware, high-level application logic can be added functionalities and framework around it without changing the application logic at all, and applications can adapt to different hardware configurations without changing any internal logic.

In Intel-NTU Center Special Interests Group for Context Analysis and Management (SIGCAM), our team have been collaborating on a project, called WuKong, to develop an intelligent middleware for developing, and deploying machine-to-mahicne (M2M) applications. The main contribution of this project is to support inlligent mapping from a high-level flow based program (FBP) to self-identified, context-specific sensors in a target environment[13].

1.2 Problem definition

1.2.1 Component Based Fault Tolerance System

The development and deployment for a fault tolerant application is still immature in most component based middleware. Even though components are modular in providing reconfigability to applications, they are still not failure resistent and cannot recover from failures. The problem is worsen when the number of components increase in applications, the developers would still bear the burden of manually programming the applications to ensure fault tolerance.

1.3 Proposed Solution

I propose to investigate how applications described in high-level flow based program language could translate to low level constructs to create a fault tolerant, applications in WuKong. In detail, we investigate how system components collaborate to achieve a common goal while satisfying application requirements to achieve self-fault detection, self-fault diagnosis, and self-fault recovery.

With colleagues from the Intel-NTU Special Interests Group for Context Analysis and Management (SIGCAM) at National Taiwan University, I have developed a new intelligent fault tolerance system, called Fur (temp), as part of WuKong project, an Intelligent Middleware for developing and deploying applications on distributed platforms. Our proposed system consists of agents collaborating to simplify fault tolerance development and to shorten deployment cycle for heterogeneous M2M applications.

The frivolous nature of requirements in applications, and actual physical sensor environements, along with the hard to predict user priorities, each contributes a unique challenge in its flavor to developing an adapatable fault tolerance solution.

Below are the list of areas that this work will address solutions in.

Intelligent Mapping

Flow-Based Programming (FBP) Paradigm has been used in WuKong to enable loosely coupling between high-level application logic and low-level hardware constructs. WuKongs also achieves late-binding to bridge the worlds together at the last stage of deployment using a technique known as intelligent mapping. Intelligent Mapping is a process in which high-level application logics are broken down into components and then mapped to appropriate nodes. Sensor Profile Framework (SPF) are used in mapping to handle heterogeneous sensor platforms, thus applications can be successfully converted into lower hardware constructs to generate low-level intermediate code to deploy to the sensor network.

Sensor Profile Framework

Sensor Profile framework provides an high-level abstraction to sensor capabilities to enable building more complex application logic.[13]

User Policy Framework

Allowing user-friendly specification of application executive objectives, and context-dependent management of system peformance.

Group Communication Systems

A group communication system deals protocols for synchronizing group states among group members in an consistent manner.[4]. When the applications are deployed to the sensor network, groups will be formed to implement redundancy. For a group of low-power sensor to collaborate on a common goal, along with high-level application requirements and Sensor Profile Framework, a new group communication protocol is needed to support fault tolerence.

1.4 Thesis Organization

Our work overlaps many diverse but interconnected domains, each topic being itself a subject of advanced research and abundant literature. The second chapter gives a brief background overview of these domains. We start by describing wireless sensor networks. Then we go on to discuss fault tolerant design for distributed systems, it's objectives and recent developments. Finally, we will be talking about component model based middleware. The third chapter gives some overview of related work. The following two chapters describe our work in fault tolerance for WuKong system. We first give an overview of some essential components in WuKong. Then we give a comprehensive description of our fault tolerance system architecture. We conclude this chapter by highlighting how the integration of those subsystems, through careful scrutiny, could bring to the development of a fault tolerant WSN application. We then present two experiments to evaluate the performance, correctness of each mechanism in the following chapter. This thesis concludes with a summarization of our proposed system and future work.

Chapter 2

Background

2.1 Wireless Sensor Networks

Sensor nodes are equipped with low-power, low-cost, and failure-prone sensors or actuators. Sensor networks are networks of sensor nodes that connect to the physical space that are intrumented to produce data that could be meaningful for further research. They collaborate to collect, process and disseminate environmental information[1].

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 networks are nodes that communicate through air by sending electronic signals. Wireless communications aren't stable, as it is highly influenced by environmental factors.

2.1.1 Redundancy

Sensor networks are usually deployed in large scale and unattended in long period of time. Sensor networks communicate with low-power wireless radios to aid scientists in collecting spatial data that could lead to more understanding of the environment. However, several challenges such as node failures, message loss, and sensor calibration leaves the effectiveness of sensor networks in question. With the assumption of spare homogeneous resources, redundancy is used in sensor networks to increase fault tolerance against node failures. The system is designed with backup nodes that could automatically recover and replace should one node fail.

2.2 Component Based Middleware

Middleware enables communication and management of data that simplfies complex distributed applications.

As most applications for wireless sensor network involves management of data and communication between network of nodes, middleware is integral in providing a unified experience for implementing more complex architecture such as service-oriented architecture.

However, the separation of design abstractions between low-level hardware and high-level application logic has not been successful in sensor based systems.

It is also not successful in terms of making them adapatable and evolvable for new services in new environments.

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.

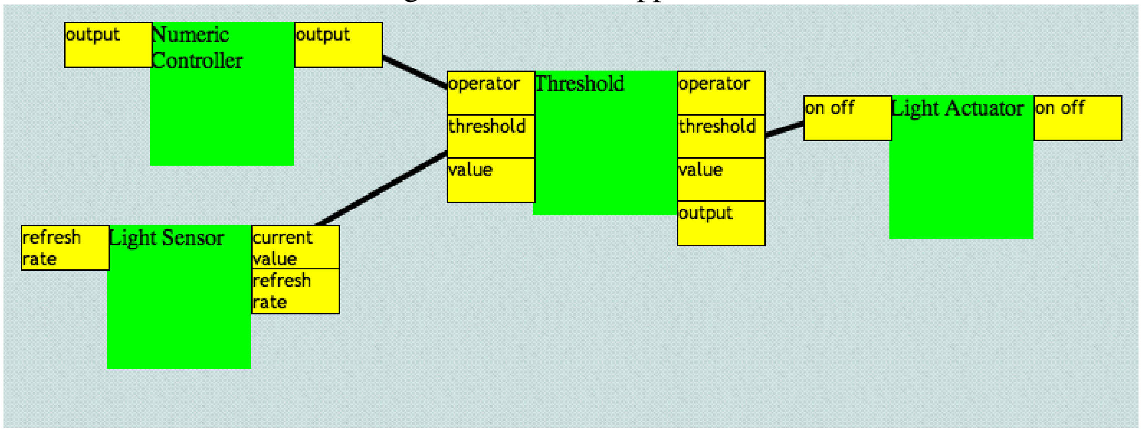
In Intel-NTU Center Special Interests Group for Context Analysis and Management (SIGCAM), we have been collaborating on a project, called WuKong, aiming to develop an intelligent middleware for developing, and deploying machine-to-mahicne (M2M) applications with ease. The main contribution of this project is to support inlligent mapping from a high-level flow based program (FBP) to self-identified, context-specific sensors in a target environment[13].

2.3.2 Flow Based 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 is best suited for describing M2M applications as it allows the developers for the applications to focus more on the abstraction meaning of the com-

Figure 2.1: A FBP application



ponents 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 2.1 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 greenblocks 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 a sensor, or an actuator while some other components could represent virtual processes such as mathmatical 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 is only made with properties with matchinkg data type. The FBP applciation in Figure 2.1 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 deteremined during deployment, that it is represented by on or off.

2.3.3 Sensor Profile Framework

While FBP defines the logical view of an application, WuKong 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, it 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. Drawing from the example in Figure 2.1, 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 WuClasses. It allows 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 inputs.

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

Property Propagation

The profile framework is in charge of communication between WuObjects as well, which are not necessarily on the same nodes. Profile Framework monitors the changes in properties and propagate 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 Profile 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 2.2: WuKong application build flow

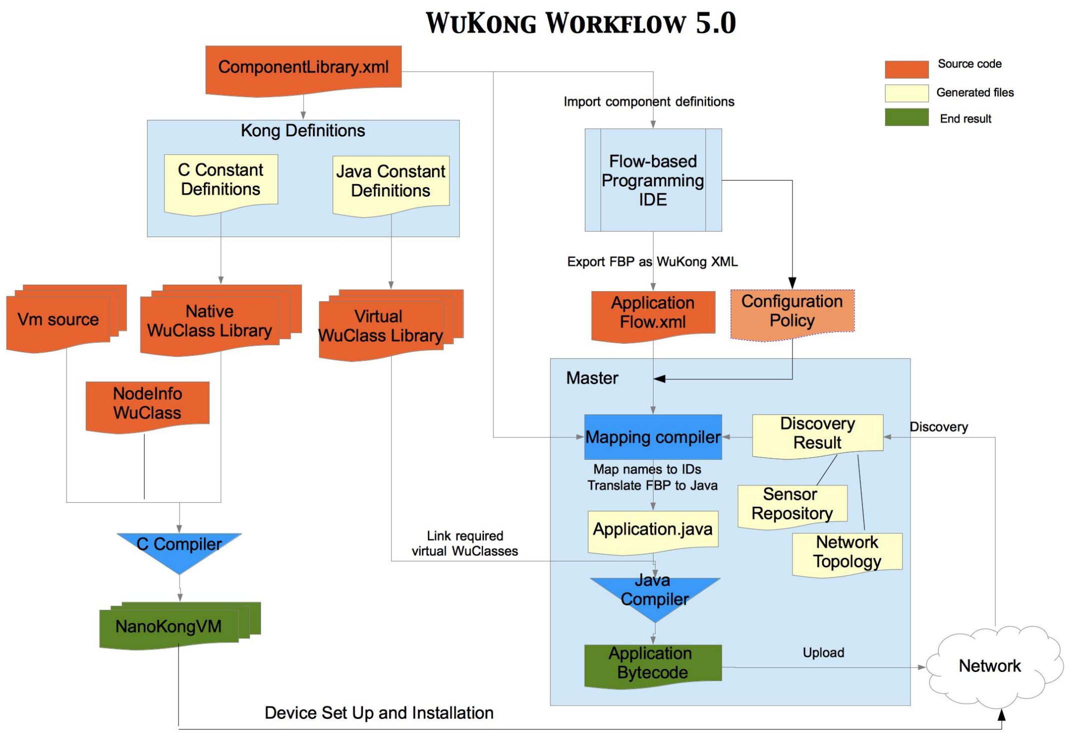


Figure 2.2 shows the overview of WuKong's build flow. The left part represents the build process for NanoKong VM which will be installed on the sensor nodes as part of the WuKong framework. The top part represents the build process for generating component libraries and Virtual WuClass library which 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.

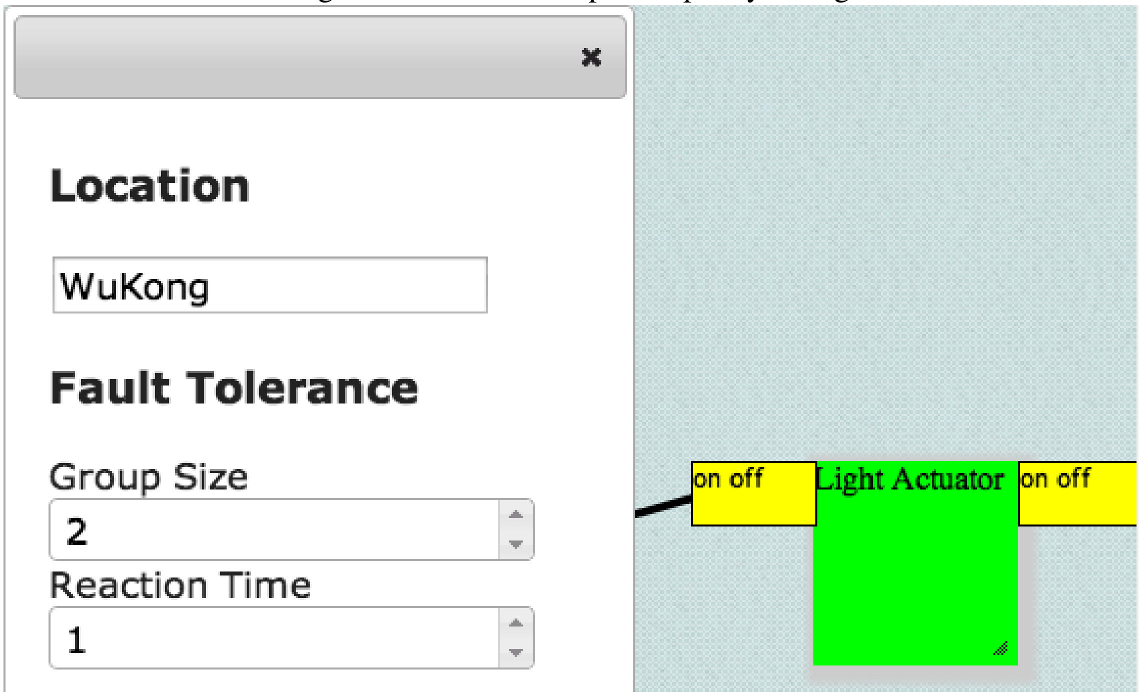
The Java code consists of many parts from different phases of the mapping process. First, the Java code contains information about links between components that were taken from the FBP XML passed in earlier from the IDE. A link contains the source component id, destination component id. The library code 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 substitude 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 bluebooth 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 wireless sensor networks.

2.3.6 User Policy Framework

Figure 2.3: A user component policy dialog



Location

WuKong

Fault Tolerance

Group Size

2

Reaction Time

1

Many M2M applications are heavily influenced by user preferences and current environmental context, as users and objects are mobile and application requirements and policy could change in time. Users are able to specify user policy for every component in the FBP and the application as a whole as illustrated in Figure 2.3

Fault Tolerance policy

M2M applications are inherently distributed, and hence it is inherently prone to failures since all nodes are running autonomously unattended for a long period of time where the external enviornmental influences could break and shut down the devices easily. Fault Tolerance policy enables users to specify relevent policy for tolerating failures in the granular level. This thesis will discuss more on fault tolerance policy in the following chapter.

Chapter 3

Related Work

3.1 Component Based Middleware

Distributed Systems such as Wireless Sensor Netweorks (WSNs) composed of multiple resource constrained devices equipped with low-power radios, low-cost sensors collaborating to perform tasks for applications in multiple discipline such as habitat monitoring. However, typically WSN applications are huge in scale, subject to unreliable network, high risk of node failures, and expected to run unattended for a long period of time, several work on lightweight component models have been proposed[5][7][6].

3.1.1 Lightweight Component Models

Work such as LooCI by Hughes et. al.[9] and Remora by Taherkordi et. al.[16] have demonstrated the feasibility of reconfigurable infrastructure with event-driven programming model, event management system, dynamic-binding to reduce development overhead, simplify abstractions, and easier, dynamic reconfiguration for distributed applications.

3.2 Fault Tolerance in Distributed Systems

Many systems has been proposed to provide fault tolerance in distributed systems, as the possibilities partial failures is a fundamental characterisics of distributed systems. However, distributed systems such as Wireless Sensor Networks typically involves a large number of resource-constrained devices equipped with various hardware components such as microprocessors, sensors, memory, wireless communication. An application rely on the collaboration among the sensor nodes in the network to perform tasks.

In order to prevent a single point of failure in applications that would bring application to a halt in the event of failure, one of the primary goals for most of the work for fault tolerance is to eliminiate single point of failure in an application, so replications, redundancies are recognized to be a good model for tackling the problems mentioned earlier in distributed systems. Replication and redundancy are both techniques that allow the system to duplicate multiple copies of a specific system component such that in the event of failure of one of the components, one of the other duplicated components can take over to perform the same services or functionality that the previous one provides.

3.2.1 Service-Oriented Architecture

Neumann et. al.[10] proposed a new redundancy infrastructure to bring Service-oriented Architecture (SOA) to Wireless Sensor Networks (WSN).

SOA is an established approach to ease development of complex distributed applications by encapsulating system compoennts into services, this allows a more flexible way to construct and develop interactions in application.

However, these approaches don't work well with applications where reconfigurability, interoperability, and heterogeneity are requirements. Existing approaches do not consider reconfigurability in the application level that might invalidate existing infrastructure built upon one instance of applications in another new instance when redeployed. Most existing approaches also are not efficient in providing reducing the development overhead to simplify fault tolerance in existing applications.

Chapter 4

Fur: An Intelligent Fault Tolerance System

This chapter describes how subsystems in Fur collaborate to provide fault tolerance for applications. First we will describe how users could specify fault tolerance policy for the application, and then how WuKong Master take the policy, application and discovery results and compile into low-level intermediate representation that could be deployed to the network. Lastly, we describe how the sensor network detect, recognize, and recover from failures autonomously based on the result of the mapping.

4.1 WuKong Applications

?? illustrates a typical WuKong application with components connecting to form a small network. What this figure shows is that applications are made of components, and components are connected through properties. Any pair of properties form links that binds components together. Connected components interact with each other based on the direction of the connection. The properties on the left of a component are inputs to this component, to the right side are the outputs of an component. Thus one can infer that a connection cannot connect to both inputs or outputs of any pair of components as both of them have the same data flow.

4.2 Fault Tolerance Policy

In this work, fault tolerance policy can only be specified on the level of components, so there is a separate policy for every component. For every compoennt, users can specify two rules to guide and influence how fault tolerance works at the hardware level.

Of all policy parameters to model a fault tolerant system, there are two that are required for any fault tolerance systems in environment unique to WuKong.

1. *Minimum Redundancy Level* The minimum number of devices that will be supported as a backup for a particular component. Therefore when one of the devices fail, others will take over.

2. *Maximum Reaction Time* The maximum latency a failure will take to detect within the heartbeat group it is detected.

With the minimum redundancy level per component, each component has a guarantee that when deployed the system will be configured to support at least the level of fault tolerance for the component in the application.

With the maximum reaction time per component, each component is also guaranteed to react to a failure within the specified time (in seconds)

The policy requires the system to changes its design process to produce a fault tolerant system that can meet the requirements.

The policy for every component will be taken by the WuKong Master, along with the FBP application and discovery results, to produce the final mapped results that could be compiled into low level bytecode that would be executed on the network.

4.3 Redundancy

The primary design for fault tolerant distributed system is based on the concept of redundancy and distributed systems usually have advantages to have spare resources. Not only it is good to combat partial failures, it also provide durability to the system for an extended period of time.

Spare resources address the first fundanmental characteristics of fault tolerance that there is no single point of failure within the system.

One of the challeneges in producing a fault tolerant design in a heterogeneous network modeled by the WuKong's sensor profile framework is that devices are partially homogeneous, meaning that the hardware setup is different from node to node, so it is not straightforward to simply assign a device as a complete backup of another device.

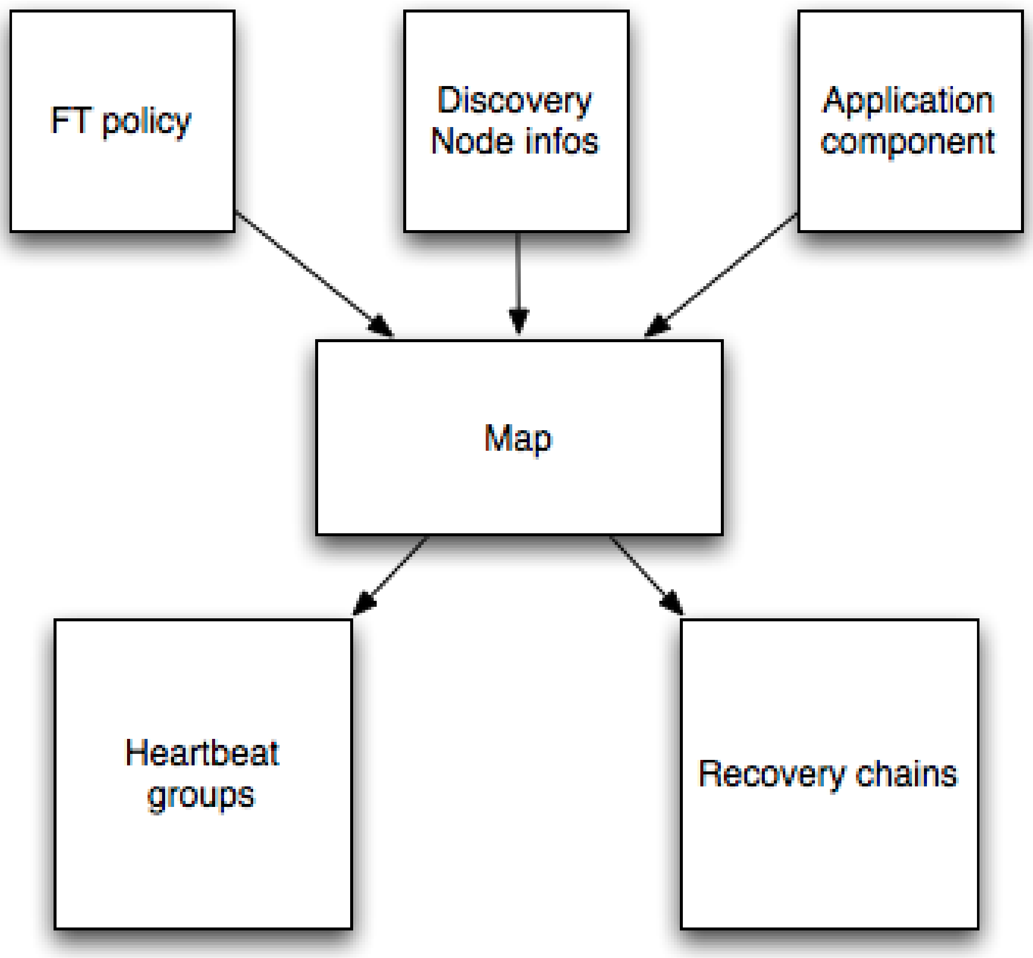
The solution introduces two new system abstractions that could address the challenge: Recovery chains and heartbeat groups.

4.4 Mapping for a fault tolerant system

The figure 4.1 above shows the elements required in the mapping process for a fault tolerant system.

Mapping is a process of converting a high level abstractions to low level bytecode representation that will be deployed to the network.

Figure 4.1: Elements for mapping a fault tolerant system



The high level abstractions are the network node infos, application graph of components and links, fault tolerance policy for components.

Mapper requires discovery of the network in terms of node infos, application graph. In order to produce a fault tolerant system which satisfy fault tolerance policy, mapper produces heartbeat groups and recovery chains necessary to construct a fault tolerant system under the unique challenges of heterogeneous sensor network modeled by sensor profile framework.

With heartbeat groups, a network of nodes will be able to detect a node failure within the network efficiently, and able to reconfigure autonomously to reach its stable state after the failure.

With recovery chains, the semantics of application will be persistent as long as there is spare parts of components that could be recovered from a node failure and still meet its application requirements.

4.4.1 Heartbeat Groups

Heartbeat is a heuristic to detect failure. Our work assume a fail-stop model where sensor nodes fail by not sending messages for a period of time. However, in a group of nodes, heartbeats needs to be arranged in such a way that no failures could be left undetected. And thus Mapper will have to determine the heartbeat arrangement, dependencies to ensure every sensor node is monitored and will not fail without notice.

Heartbeat arrangement usually depends on the applications and sensor arrangement [8], thus most related work aim for a specific type of application and sensor arrangement to simplify heartbeat arrangement, however since application type and sensor arrangement in WuKong are not known until deployment, therefore a new way to arrange heartbeat has to be devised.

In WuKong, applications are drawn on a canvas which consists of blocks that gives the type of sensors or resources needed and lines which connects the resources together in such a way that would satisfy the requirements. However, arrange heartbeat based on the flow of the components is not desirable since applications are not the immediate representation of the underlying network topology such that some components could be mapped on the same node, and some connected components could be mapped to nodes that are far away from each other that require multiple hops to reach. Arrangement based on such high abstractions would incur extra communication overhead by making a lot of nodes as a relay for heartbeats.

The proposed solution will produce application agnostic arrangement and at the same time be used in producing the arrangement of the components in the network to reduce the communication overhead as much as possible.

The algorithm assumes that links are symmetrical and toplogy is stable after making the arrangement.

Heartbeat groups are clustering of nodes that are one hop distance to each other, in other words, fully connected. By grouping nodes within one hop distance together, we significantly reduce the possibility of heartbeat hopping which is a big factor in communication overhead.

Heartbeat groups are determined by picking an anchor node by random in the nodes, and then iterating through network of nodes to cluster nodes that are meeting the criteria (one hop distance) and repeat again until all nodes are exhausted. The function to determine topology of the network comes from the hardware of the wireless radio specifications.

It is possible that due to some peculiar network topology that the algorithm would produce single node group which itself could not be of any use.

If that happens, the algorithm will be rerun again. Since the anchor is picked randomly from the remaining node list, the algorithm will not produce the same result thus eliminating the problem.

Once the groups are formed, each group could employed many different heuristics to connect such that any failure could be detected. One of the heuristics that is used in this work is called daisy chain. Daisy chain makes every node monitoring one other unique node in the network such that every node is monitored with minimum communication overhead and an even communication load for every node.

4.4.2 Recovery chains

When a minimum redundancy level for a component is higher than one, WuKong Master would find and map at least amount of eligible nodes as candidates to carry the WuObject coresponding to the component in case of a failure.

When a failure is detected, the detector is not necessarily carrying or knowing what resources there are in the network at the given moment, and it would produce an enormous of overhead to query for the resources in the network, given our scenario constraints with tiny sensor nodes, we have to reduce as much overhead as possible.

Our current approach employed a technique that make the order of the candidates significant such that when a node failed, the next one would take over, and since it could be stored as an list, the information could be serialized and stored on the nodes to reduce communication overhead the detector would have to do whenever a failure is detected.

Candidates are first filtered by WuClasses that correspond to components application requires from the list of nodes discovered, which are eligible to provide certain resources for components specified in the application. Once it is sorted out, the algorithm will produce a dictionary of component id as keys and list of node ids as values.

Every WuKong node could be queried with node infos, a node info constitudes information such as a list of WuClasses and a list of WuObjects that are used to help Master greatly in making an informed decision to construct a list of eligible candidates for components.

The system will warn against the users if any component candidates does not satisfy minimum redundancy level policy for the component.

Sort Candidates

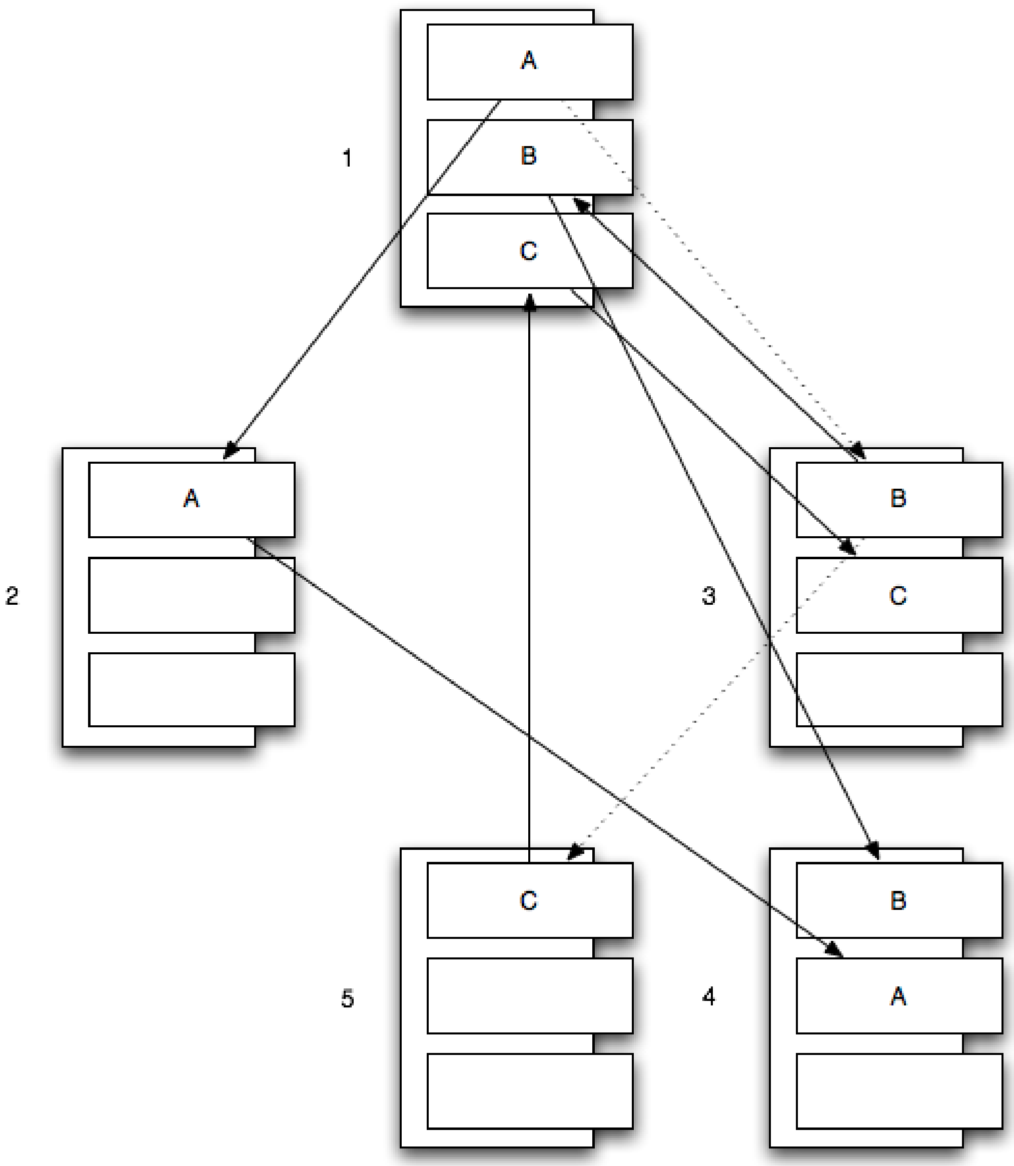
There are two types of connections between components in the network depending on how and where they are situated. If components have a link and are situated in different nodes, it will be message passing, but if components have a link and are placed in the same node, it will be function call. One of the ways to minimize communication overhead over the course of time over node failures is to maximize function calls and place linked components as close as possible.

Leaving the order of candidates as it is might have a negative effect to the system that while it could recover from failures, the new assignment is not optimized, and thus it will produce unnecessary communication overhead which might compromise the operation of the system. However, if the order of candidates is carefully analyzed for every possible failure scenario, making sure at any point in time, the system is in its optimized state, the system will have less chance running into undesired states.

To illustrate this even further, the figure 4.2 illustrates a scenario of an application deployment where candidates of components are placed without taking account of communication overhead, the solid arrows are the recovery chain for the components, and dashed arrows are the links between active components. There are 5 nodes numbered from 1 to 5, and each carries components. The components active are the onces linked with dashed arrows. The application specifies a link from component A to B and B to C, and component A on node 1 is connected to the component B on the node 3, and component B is connected to component C on node 5.

A better placement would be to put all component to node 1 thus all links are function

Figure 4.2: Undesired candidate mapping scenario



calls thus reducing communication cost dramatically, also when node 1 fails, node 3 will take care of the links between component B and C it would still retain and have the minimum communication cost.

And when node 5 failed, component C will elect node 1 to take over, but that wouldn't be optimal since the link to to component C from component B is in node 3. If the system chose node 3, the link between B and C will be internal thus reducing the communication overhead. Component A cannot be further optimized. Further more, if all components elect node 1 to carry at certain point in time, it will be the most optimized since none of the links are external.

The algorithm is able to solve two problems, first it will find an optimial mapping given current network topology, after mapping with lowest communication overhead in subsequent failures.

Algorithm 1 Sort Recovery chains

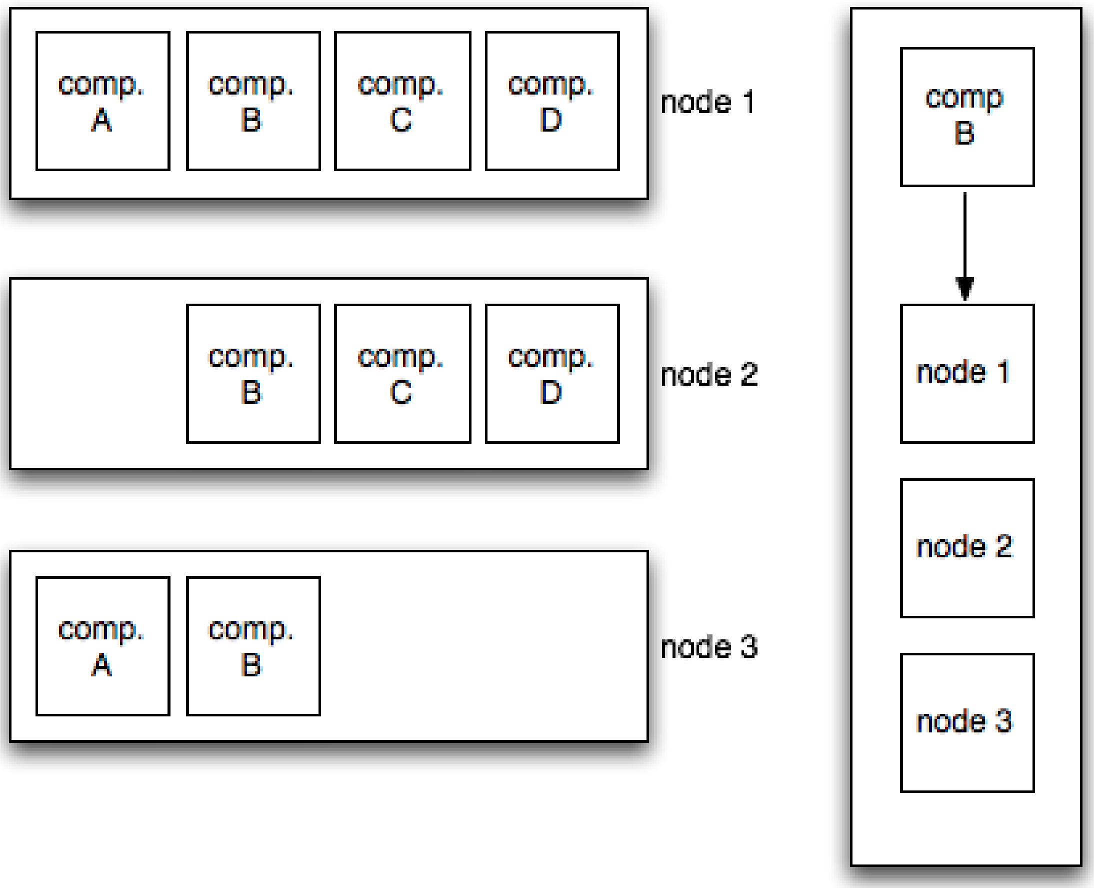
```
Require: A list $C$ of components
Require: A dictionary $D$ of unsorted recovery chains of every component
Require: A histogram $H$ of the occurrances of nodes appearing in components
Ensure: A dictionary $D$ of sorted recovery chains of every component
  for Every $c$ in $C$ do
      Sort $D[c]$, and use $H$ as keys in decending order
  end for
```

Given a list of nodes with any number of component slots (WuClass), for any node that fail, the result network topology could leave the new assignment unoptimized that would produce unnecessary or avoidable communication overhead in the form of message passing between links.

This algorithm could order candidate lists such that it will produce an initial mapping with minimal communication cost and also for any network topology due to failures in the network as long as no components have their candidates running out.

The algorithm 1 takes a list of candidates for all application components and sort them based on the number of unique component slot (WuClass) that a node could carry. So if a node can carry 5 unique components, it will be sorted in all component candidate list

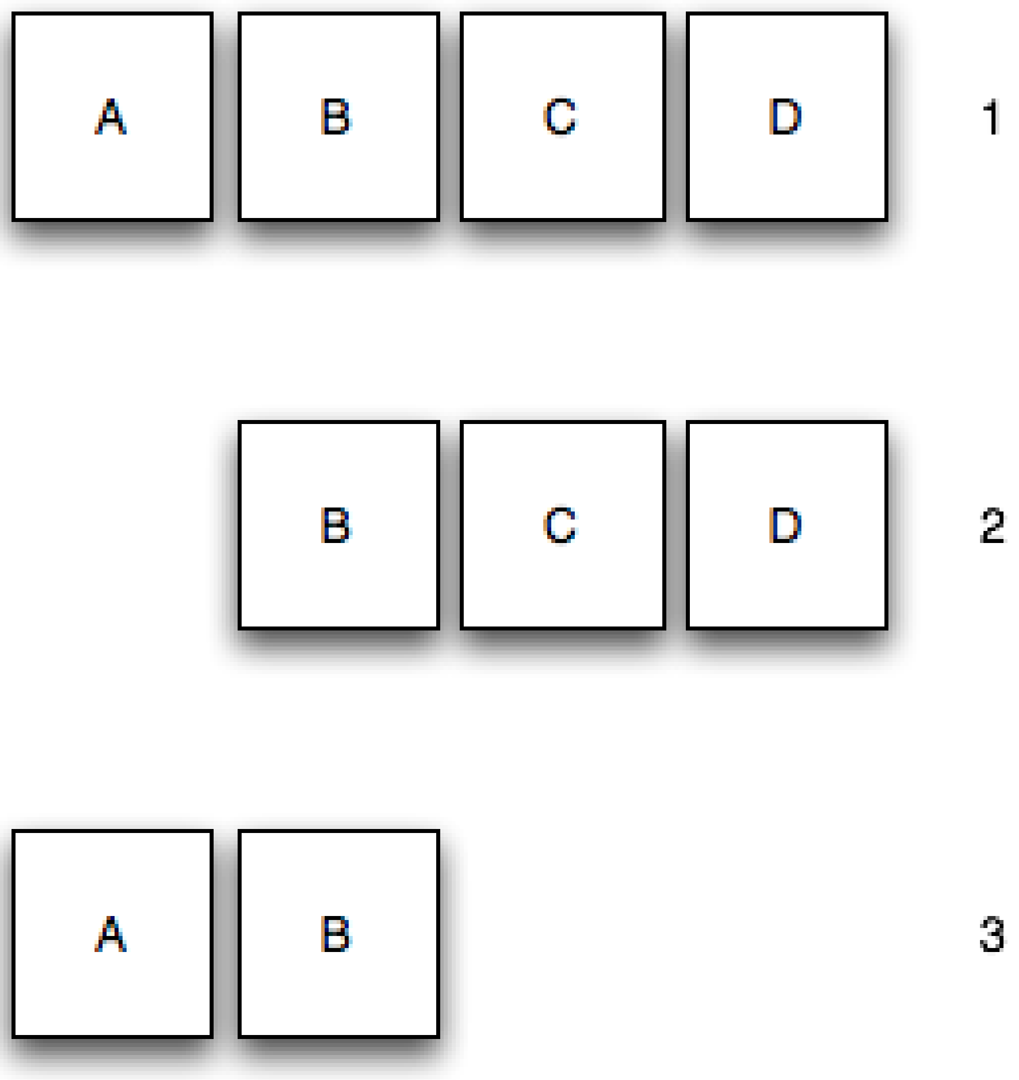
Figure 4.3: An example of a sorted candiate for component B



before other nodes that could contain 4 unique components and below. The figure 4.3 illustrates a sorted candidate list in action.

Assuming there is a list of nodes where each one has a list of WuClasses that it could support. Sort nodes based on the number of WuClasses it contains, and align and sort Wu-Class lists so that the same WuClass will be aligned vertically as illustrated in figure 4.4. Assuming there are three links in the application where component A connects to B, B connects to C and where C connects to D, one can observe that the best assignment for the network present in the figure is to elect node 1 for all components so all links will

Figure 4.4:



be internal, thus there is no external communication overhead, which is more preferrable than electing node 2 or node 3 as first candidate. But let's assume that there exist a better mapping which nodes with less unique components is preferred, if that's true that means excluding the links which would be external in both choices, the links in the less unique component node somehow has internal links even though the components aren't residing on the same node, a contradiction, thus nodes with more unique components is preferred and will have less external links. And once the node with most unique component dies, it could be reduced to a smaller version of this problem and be solved again in the same way.

4.4.3 Determine Heartbeat Group Period

Once the placement and ordering for the components in forms of recover chains are decided, the heartbeat interval for heartbeat groups could be deteremined. Since heartbeat groups could have multiple components, the group heartbeat period is half of the minimum maximum reaction time policy of the components.

Our system assumes a fail-stop model where nodes are suspected dead when it does not send out messages, but to compensate possible deviation in communication latency, a failure is detected when the node does not send a heartbeat message within two normal heartbeat periods. Thus the heartbeat period is half of the reaction time.

Algorithm 2 Determine Group Heartbeat Period

```
Require: A list of heartbeat groups $H$
Ensure: A list of heartbeat groups with heartbeat periods
  for Every heartbeat group $g$ in $H$ do
      for Every component Component($n$ in $g$) do
          Find the lowest minimum reaction time
          Divided in hald and assign the period to group $g$
      end for
  end for
```

Since all nodes within the heartbeat group is compared against, so only the lowest time will be picked.

4.5 Information Distribution

After mapping, the information of recovery is distributed accordingly based on the heartbeat group results to ensure the integrity of recovery process such that the detector will be able to recover broken links and update the nodes that are affected.

After mapping and produced heartbeat groups for the network and recovery chains for the components, WuKong would have to generate and assignment information appropriated to every node to support fault tolerance.

Every node will have information pertain to the list below:

1. Recovery chains for the mapped components
2. List of nodes of the heartbeat group it is in
3. The recovery chains of the node it will monitor based on the heartbeat arrangement
4. The application links of the components assigned to the node it will monitor based on the heartbeat arrangement
5. Heartbeat period of the group it is in

4.6 Application generation

WuKong runs a JVM on every node in the network, the VM provides services for the application program in Java to access lower parts of the resources. This section will introduce the list of newly added interfaces and resources to support and enable fault tolerance for the application.

1. component instance id to WuObject address map

2. heartbeat groups member list

3. heartbeat group period list

4.7 Wireless Deployment

The rest of the deployment is the same as the deployment described in the background work for WuKong, where after it generates the Java code, it compiles down to lower byte-code representation and send it to every node wirelessly. The nodes will be reprogrammed by taking the code and reload in their flash memory.

4.8 Fault Tolerance System

4.8.1 Towards Failure Detection

As we mentioned earlier why sensor system has to evolve to adapt to crutial, ever changing environment, one of the first things a system could achieve that goal is to detect failures autonomously.

Heartbeat

A failure in distributed system could come from different causes. Some nodes might fail because of software bugs; some nodes might appear to fail because of poor wireless link quality. One of the biggest challenges in distributed systems is to be able to detect failures when it occurs accurately.

In distributed system, nodes could fail. In order to be consistent, we model failure by whether a node send messages within predefined period or not. It is called a fail-stop failure model.

Heartbeats are messages sent by individual nodes periodically to indicate to the monitoring nodes its health.

Figure **??** illustrates some nodes sending heartbeats that detects a failure when a number of expected consecutive heartbeats have not been received.

In our model, a node is considered dead when it has not been sending heartbeats for more than 2 timeout periods.

Decentralized Detection

In a network of nodes, the arrangement of heartbeats has to be addressed to have full coverage of the network such that every node is monitored by another node which is also monitored by another node and is decided during the mapping phase mentioned in the previous section.

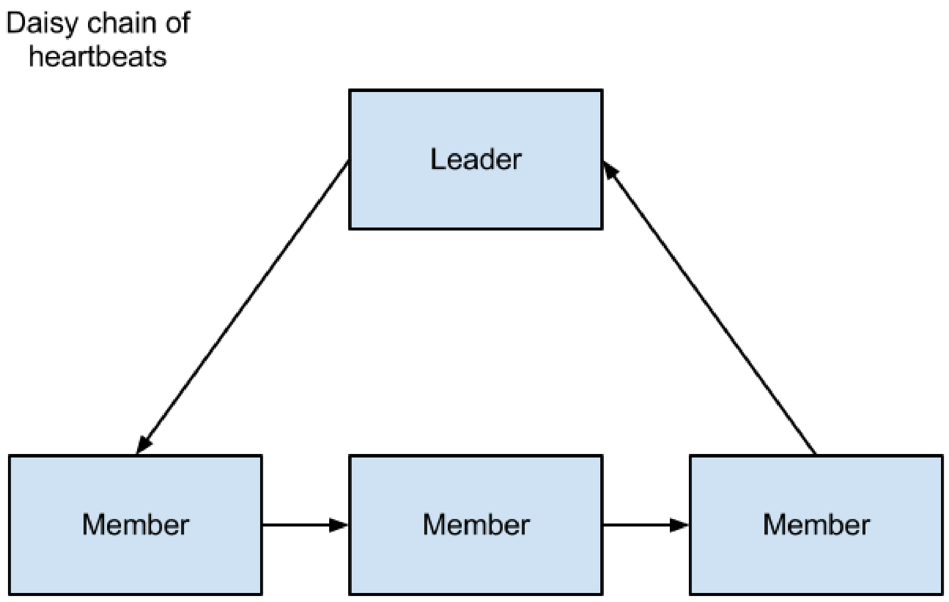
Centralized detection makes one special node to monitor every node in the network by making them sending heartbeats to it. The arhictecture has several weak points where it requries extra resources to monitor, and since the monitoring node cannot be replaced when failed, the system will have single point of failure. In addition to that, the average communication overhead for the monitoring node will be very high and saturated since all the traffic goes towards one place and thus the system performance will deteriorate.

A heartbeat can only monitor one node at a time, so in a network of nodes, we need a heartbeat network. The structure of heartbeat communication pattern highly depends on the underlying network assumption and infrastructure of the application. To produce the most efficient network with the least connections, heartbeat network has to satisfy two properties.

1. Every node has to monitor at least one node other than itself
2. Every node can only has one node monitoring itself

A heartbeat network in the form of a daisy chain is one of the networks that satisfy both properties as shown in 4.5. Every daisy chain heartbeat network monitors all nodes in the group, given by the properties that every node has to monitor at least one node and every node can only have one node monitor itself. So if there is n nodes, there can be at most n nodes being monitored, every monitoring node can only monitor one node, every node can only monitor node that others have not, thus every node is monitored by only one unique node. Thus this daisy chain guarantees every failure can be detected.

Figure 4.5: Daisy Chain of heartbeats



Message complexity Every heartbeat takes one message to send from one to another. As of current, we assume every heartbeat is sent using unicast, and there is no ACK for heartbeat messages. The message complexity for standard one-hop star network takes about O(2n-2) messages since the leader sends n-1 to every member and every member would also has to send a message back to leader, that comes to double of the single traversal from leader to other members.

The message complexity for the daisy loop takes about $O(n)$ messages for a group, because every member only sends one heartbeat to one other member at a time including the leader.

Heartbeat Group Recovery

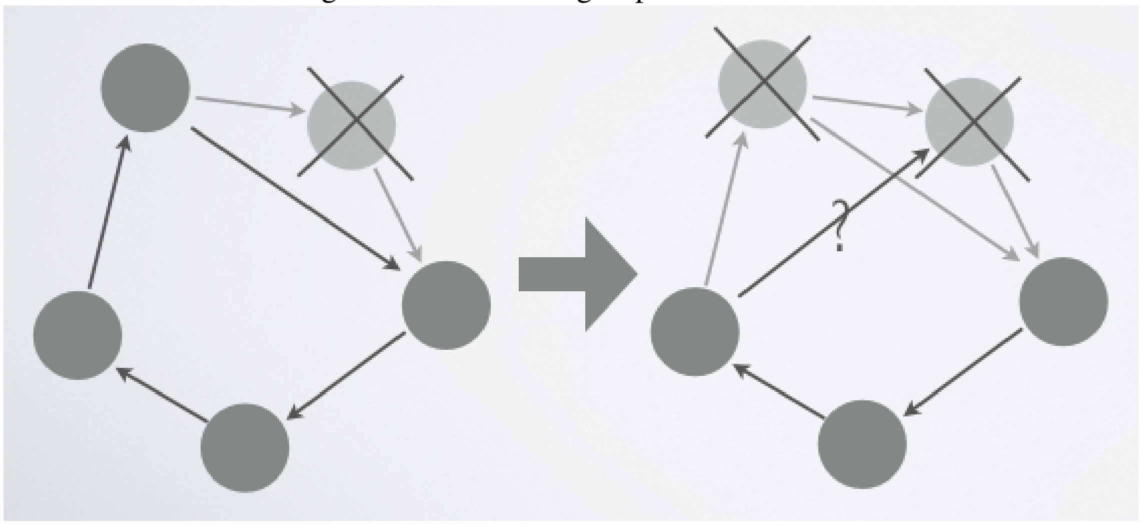
When a node failed and stopped sending heartbeats, the existing network of heartbeat needs to recover its arrangement to handle the next error. If we are given a network of nodes with heartbeat arranged like in Figure 4.5, when the leader failed and is picked up by the member, the member that is monitoring the leader will notify the member that is monitored by the old leader to redirect its heartbeat to itself instead so the loop hole is closed. Algorithm 3 described what happened when a node detects a failure.

Algorithm 3 Recover Heartbeat Group (at detected node)

```
Require: A list of node ids $G$ of the heartbeat group it is in
Require: Failed node id $f$
Ensure: Update list of node ids $G$ of the heartbeat group in other members
  Remove failure node id $f$ from local Heartbeat Group $G$
  Update the new node id to monitor which is the node before its id in the list
  for Every $n$ in $G$ do
      Send message to $n$ to upload their local heartbeat group and update their monitoring
          node id
  end for
```

The algorithm requires synchronization among all heartbeat group members since if it only updates the affected nodes then after two specific node failures in such order that the node will not be able to recover from failure. It is illustrated in Figure 4.6

Figure 4.6: Heartbeat group cannot recover



In the first failure, the previous node could update its local list point to the alive node to monitor, but if the node fails, the next node will not point to the alive node to monitor.

4.8.2 Failure Recovery

Due to the nature of the network that will have lots of recovery chains for several application components on different or same nodes, when a node detects failure in a node, the detector will initiate a failure recovery protocol to recover the affected component recovery chains and application links for nodes hosting connected components.

The detector of node failure will determine the WuObjects running on node, to update recovery chains in the network, and to determine nodes that host connected WuObjects. For example, if node A runs WuObject 1, and when WuObject 2 which is hosted on node B is connected, node B is the node hosting the connected WuObjects.

Recover Chains

The algorithm 4 describes how the node detected the failure will have to update recovery chains of all nodes that is hosted on the dead node.

Algorithm 4 Recover Recovery Chains (at detected node)

```
Require: A list of components $C$ of the failure node
Require: Failed node id $f$
Ensure: Update all host hosting component in $C$ their local recovery chain lists
  Remove failure node id $f$ from all local recovery chains of component in $C$
  Update nodes of their recovery chains in the components in $C$
```

Connected Nodes

The algorithm 5 describes how the node detected the failure will have to update application links of all nodes that is connected to components on the dead node.

Algorithm 5 Update Application Links (at detected node)

```
Require: A list of components $C$ of the failure node
Require: A list of links of components $L$ of the failure node
Require: Failed node id $f$
Ensure: Update all local links in $L$ to point to next alive node for the component with the dead node
  Update links of nodes that host components that is connected to the original component with the dead node
```

Chapter 5

Experimental Design and Result

5.1 Experimental Setup

All boards are equipped with an Atmel ATmega1280-16AU 8-bit microcontroller with 4K of EEPROM and 64k of flash. The boards hardware design is based upon Arduino hardware referenced design, in addition, every board has wires for mounting multiple wireless protocol adapters such as ZWave, ZigBee. In the following experiments, every board is only equipped with a ZWave adapter, and only communicating through ZWave.

Every board is also pre-installed with a modified version of NanoVM (ref here) called “NanoKong” that supports all the basic WuKong framework protocols including the new additions from the work in the previous chapter.

A PC with wireless access is dedicated for hosting the WuKong Master software which is responsible for managing WuKong applications for the whole system and serves as a mean to present an interface to the users.

Three boards will be used in the experiments below. One of them is equipped with a light sensor that returns a byte indicating the light level around the sensor. The rest are equipped with a relay which each controls the power supply of a lamp.

An additional board with the same hardware specification is used as a gateway between the Master and the sensor network.

5.2 Experiment - Application Deployment

To evaluate the performance of the result of the new mapping algorithm and the overall fault tolerant framework, an application shown in Figure 1 will be deployed.

5.2.1 Experiment Design

In this experiment, an application shown in figure 1 above will be deployed with a fault tolerance user policy shown in the figure 2 upon the hardware setup described in the previous section three times to evaluate its performance.

The application requires a light sensor and a light actuator. The light actuator will turn on the light if the sensed light value is below a threshold specified from the numeric controller. Numeric controller is fixed on a value of 200. The light value takes a byte ranging from 0 to 255. The comparison is done with a virtual component Threshold in native implementation.

We will simulate a node failure by unplugging the power supply of the active light sensor node on all tries.

5.2.2 Experimental Result

The performance of the fault tolerance system is evaluated with the metrics listed below:

1. Correctness, whether the system is configured to do what the mapping result specifies, including the heartbeats, heartbeat periods, recovery chains, application links.
2. Communication overhead used for heartbeats
3. Communication overhead for failure recovery
4. Failure detection success rate
5. Fault recovery success rate
6. The average time to recover from the time of failure
7. The average number of messages used to recover the system

Mapper

The mapper will produce a mapping result, including the recovery chains and heartbeat groups, and will generate the Java bytecode that will be deployed wirelessly to the sensor network.

Fault Tolerant System

The system will be configured to send heartbeats according to the result of the mapping and recover based on the algorithm once a failure is detected.

Chapter 6

Conclusion

6.1 Summary of Contribution

6.2 Future Work

6.2.1 Questions

Q: How does the system recover from two consequtive failures in the network?

A: We assume a much simpler failure model where there can be at most one failure happen at any point in time. It is a reasonable assumption given that our system recover from failures fairly quickly (2 secs) ....

Q: How does an application is being compiled into low level bytecode? And when and how does WuKong reconfigure the nodes?

Q: What happen when an application is deployed? What is specifially at work?

6.2.2 False positive fault detection

It appears to be possible to have false positive fault detection when a node is not dead but actually got partitioned away from the network for a short period of time. If it is the leader that got partitioned away for too long, several members will be detecting this failure and they might all initiate a leader election. Since they all know of this situation, every node detecting a failure will wait for a random amount of time before sending the message. If a leader election message has been received, it will terminate its current action and continue to the second phase of the leader election process.

However, it is possible that leader is not actually dead, and it is also monitoring the members. The leader might conclude that the members are all gone and will also generate a failure event (since there is no one to synchronize to). This is a split brain problem because the remaining members will elect a new leader and proceed in synchronizing the link table in neighbor nodes, but the old leader is still operating and sending data between the neighbor nodes, this will create a conflict both in the group and cause a confusion among the outsiders.

Assuming both partitions can talk to the neighbor nodes with objects connected to their objects, there is no way for the partitions to detect the problem within themselves but only the outsiders.

The outsiders, whose objects are connected to the group, will be the fault detector and will notify both leaders of their existence along with their scoring. The leader with less scoring will give up their leadership, and try to merge with the other partition if possible. If it is still not possible after a timeout, it will try to notify the Master of this situation.

6.2.3 Group connection types

6.2.4 Mapping

Q: Inclusion problem, how to solve it? How severe it is?

A: The problem is due to the low redundancy level of a compoennt which is mapped to a node with other compoennts which have higher redundancy levels. This problem does not always have a solution, it depends highly upon the resources in the network, it is unavoidable. However, we are considering possible solutions to address this problem in the future by making sure the mapping results does not have this pattern and could generate possible solutions to solve this problem by either increase the redundancy level of the vulnerable component or purposefully avoid mapping to dangerous nodes that would put the component to risk.

But the bigger question is, if compoent FT policy is enough to ensure application durability, at the current state, if any component does not have minimum redundancy level of 2, that application is in the risk of single point of fialure, there would be needed a way to inspect whether the policy is fault tolerance at the application level, or introducing several application level FT policy and that is something that we could be looking into in our next research work.

6.2.5 Policy

When application are getting complex full with features and configurations, it is important to have a high level declarative configuration policy language to specify the control for features and control of their respective behaviors smoothly. I propose a high level policy for fault tolerance that could be translated to low level application requirements.

Bibliography

[1] V. A. Archana Bharathidasan. Sensor Networks: An Overview.

[2] A. Arora, P. Dutta, S. Bapat, and V. Kulathumani.... A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, Jan. 2004.

[3] 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*, volume D, pages 43–56. ACM Press, 2008.

[4] K. P. Birman. Dynamic Membership. In *Guide to Reliable Distributed Systems*, chapter 10, pages 339–367. Springer London, 2012.

[5] P. Costa, G. Coulson, and R. Gold. The RUNES middleware for networked embedded systems and its application in a disaster management scenario. *..., 2007. PerCom'07. ...*, pages 69–78, 2007.

[6] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A generic component model for building systems software. *ACM Transactions on Computer Systems*, 26(1):1–42, Feb. 2008.

[7] D. Gay, P. Levis, R. Von Behren, and M. Welsh.... The nesC language: A holistic approach to networked embedded systems. *Proceedings of the ...*, Jan. 2003.

[8] S. Gobriel and S. Khattab. GroupBeat : Wireless Sensor Networks Made Reliable. *...Ad Hoc and Sensor ...*, pages 477–484, 2008.

[9] 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 *2012 IEEE 11th International Symposium on Network Computing and Applications*, pages 236–243. IEEE, Aug. 2012.

[10] J. Neumann, N. Hoeller, C. Reinke, and V. Linnemann. Redundancy Infrastructure for Service-Oriented Wireless Sensor Networks. In *2010 Ninth IEEE International Symposium on Network Computing and Applications*, pages 269–274. IEEE, July 2010.

[11] P. Padhy, K. Martinez, A. Riddoch, H. L. R. Ong, and J. K. Hart. Glacial Environment Monitoring using Sensor Networks. *RealWSN*, pages 10–14, 2005.

[12] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson. Analysis of wireless sensor networks for habitat monitoring. In C. S. Raghavendra, K. M. Sivalingam, and T. Znati, editors, *Wireless Sensor Networks*, chapter 18, pages 399–423. Kluwer Academic Publishers, 2004.

[13] 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. *Sensornets*, 2013.

[14] I. Stoianov, L. Nachman, S. Madden, T. Tokmouline, and M. Csail. PIPENET: A Wireless Sensor Network for Pipeline Monitoring, 2007.

[15] R. Sugihara and R. Gupta. Programming models for sensor networks: A survey. *ACM Transactions on Sensor Networks (TOSN)*, V:1–27, 2008.

[16] A. Taherkordi and F. Loiret. Programming sensor networks using REMORA component model. ... *Computing in Sensor* ..., pages 1–14, 2010.

[17] 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. *REALWSN 2005*, 2005.

[18] G. Tolle, J. Polastre, R. Szewczyk, and D. Culler.... A macroscope in the redwoods. *Proceedings of the* ..., Jan. 2005.

[19] G. Werner-Allen, K. Lorincz, and J. Johnson.... Fidelity and yield in a volcano monitoring sensor network. *Proceedings of the* ..., Jan. 2006.

[20] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks.