Analytical Techniques for Debugging Pervasive Computing Environments

by

Atish Nigam

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

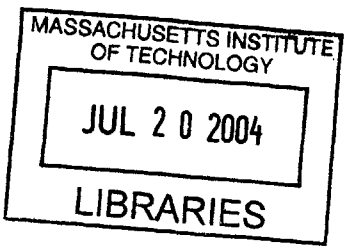
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004 [June 2004]

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by .....
Larry Rudolph
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Professor of Electrical Engineering and Computer Science
Chairman, Department Committee on Graduate Students



Analytical Techniques for Debugging Pervasive Computing Environments

by

Atish Nigam

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

User level debugging of pervasive environments is important as it provides the ability to observe changes that occur in a pervasive environment and fix problems that result from these changes, especially since pervasive environments may from time to time exhibit unexpected behavior. Simple *keepalive* messages can not always uncover the source of this behavior because systems can be in an incorrect state while continuing to output information or respond to basic queries.

The traditional approach to debugging distributed systems is to instrument the entire environment. This does not work when the environments are cobbled together from systems built around different operating systems, programming languages or platforms. With systems from such disparate backgrounds, it is hard to create a stable pervasive environment. We propose to solve this problem by requiring each system and component to provide a health metric that gives an indication of its current status. Our work has shown that, when monitored at a reasonable rate, simple and cheap metrics can reveal the cause of many problems within pervasive environments.

The two metrics that will be focused on in this thesis are transmission rate and transmission data analysis. Algorithms for implementing these metrics, within the stated assumptions of pervasive environments, will be explored along with an analysis of these implementations and the results they provided.

Furthermore, a system design will be described in which the tools used to analyze the metrics compose an out of bound monitoring system that retains a level of autonomy from the pervasive environment. The described system provides many advantages and additionally operates under the given assumptions regarding the resources available within a pervasive environment.

Thesis Supervisor: Larry Rudolph
Title: Principal Research Scientist

Acknowledgments

I would like to acknowledge the help of my advisor Larry Rudolph for his assistance, patience, and guidance during the course of our work together. His knowledge and creative ideas provided me many avenues to explore.

Through out my year the other members of the Oxygen Research Group provided me with support and good times during the past year. Albert Huang was never far away with a solution to my computer woes...I guess I never gave him a chance to run away. Debbie Wan, Nancy Kho, and Jessica Huang always provided a respite from the work day. I appreciated the constant supply of candy furnished by Sally Lee and the hilarious conversations and Texas hold'em tactics I exchanged Chris Leung.

It is impossible to acknowledge all of my friends who have helped me this past year and during my time at MIT, but I would like to acknowledge a few people who have been part of my life this past year: Alex DeNeui, the most illustrious individual I will ever let live on the floor of my room for three months. My apartment-mate Josh Juster who's penchant for gadgets provided me with an outstanding HD TV, and a superb 18th green on which to practice my putts, and for not being a hater. The man who was one step ahead of me, Ajay Sudan, I've finally caught up...now we get to go work jobs that we have absolutely no training for. I'd also like to acknowledge Pat McCaney, although he may have had issues with his sleep schedule, we were *clearly* on the same level at prioritizing our time. I would also like to acknowledge my pledge brothers in the class of 2003 at ATO. I would also like to acknowledge all those who were part of my life here at MIT for 5 years.

Finally, I would like to thank Jerry Seinfeld for producing the Seinfeld show for 9 years, and FOX for playing it every day at 7pm (and 11pm in case I was working late).

I would also like to thank my parents Anil and Reshma for giving me more support and love than I could ever imagine. Thanks for making me who I am today.

Contents

1 Introduction **11**

1.1 Background 12

1.2 Problem Description 14

1.3 Traditional Solutions 15

1.4 Assumptions About the Environment 16

1.5 Goal 19

1.6 User Interaction 19

1.7 Outline 21

2 Related Work and Background **23**

2.1 Middleware Pervasive Environments 23

2.2 Related Analysis Work 25

2.2.1 Previous Environment Analysis Work 25

2.2.2 Analytical Techniques 27

3 Transmission Analysis **29**

3.1 Transmission Rate Analysis 29

3.2 Transmission Rate Technique and Results 32

3.3 Transmission Data Analysis 39

3.4 Transmission Rate Technique and Results 40

4 System Design **49**

4.1 Health Instrumentation 49

4.2 Health Monitoring 52
4.3 health monitoring System Design 54
4.4 Other Metrics to Explore 56

5 Conclusion **59**
5.1 Problems During the Course of Work 60
5.2 Future Work 61

List of Figures

1-1 In a typical pervasive computer environment, a collection of devices, applications, and even users work together. When there is a faulty device or system, how can a user identify the source problem? 13

3-1 The transmission rate analysis calculates a *windowed* standard deviation over the past 6 arrival rate windows. Each of the boxes on the arrival rate signify an arrival rate window, the standard deviation of the past 6 windows create the entry for the standard deviation window at that time. 34

3-2 Transmission rate data for a normal run of the vnc application. The darker line indicate the arrival rate at the time window while the lighter line is the standard deviation of the past 10 time windows. 35

3-3 A graph of the average bits per second for the vnc transmission rate analysis implementation. The drop in data rate denote the time that the server became ide, the subsequent increase was when the server was made active again. 36

3-4 A graph of the outgoing rate of data from the vnc server to the vnc client in the vnc application. The darker line indicate the arrival rate at the time window while the lighter line is the standard deviation of the past 10 time windows. 38

3-5 A graph of the arrival rate of data to the vnc client from the vnc server in the vnc application.The darker line indicate the arrival rate at the time window while the lighter line is the standard deviation of the past 10 time windows. 38

3-6 A graph of the results of the distinct element counter. The estimate always yields a number less than the actual number of distinct elements. In this graph the darker lines are the actual number of elements while the lighter graph is an estimate of this using the described algorithm. 43

3-7 A graph of a normal run of the distinct element counter. The darker line is the estimate of the distinct elements sent while the lighter line is the standard deviation of this for a recent set of time windows. . . 44

3-8 This is a graph of the vnc application were a single program dominated the screen of the vnc server for a period of time and only a few screen changes were sent to the vnc client. 45

3-9 This is a graph of the vnc application that kept track of the size of elements that were sent from a vnc server. The spikes indicate times where there was a burst of data transmitted. 47

4-1 This is a screen shot of a speech enabled flash demo that allows users to send chat messages to an health monitoring monitoring server that provides information about the system. 54

4-2 A system design for a pervasive environment. The system designers provide the process monitoring components for each process. The health monitoring monitoring device then queries these systems and processes the data 56

5-1 This graph shows an overlay of data leaving a CORE device and travelling to another CORE device. The data gives indication to the causality of the transmission. 62

Chapter 1

Introduction

A fundamental problem in current pervasive computing environments is the inability to quickly and accurately identify faulting agents or devices. This thesis explores methods to identify changes that have occurred in a pervasive environment to give the user a better idea of how the environment is functioning, in order to rectify malfunctioning components.

As pervasive environments continue to be come more prevalent, it will be necessary to provide some notion of user level debugging. Systems can misbehave or act in an unexpected manner causing many problems in pervasive environments [33]. User level debugging will enable users to rectify problems created by borderline issues that are not necessarily in the scope of an environment designer's specification, but still occur. By detecting changes in a system's behavior or performance, it empowers users to fix the problem but additionally places less burden on the environment designers, as they no longer have to create an environment that can handle any possible inconsistency that may arise.

In particular this thesis focuses on an inexpensive analysis of transmission streams and the information that can be understood from monitoring the data that passes between systems, and the rate at which this transmission occurs.

1.1 Background

Modern day computing technologies have started progressing beyond the desktop to become interwoven into other parts of life [18]. Pervasive computing is the study of all aspects of these ubiquitous computing technologies, from their design to their deployment. These technologies will continue to become more popular in the coming years as computers move towards embedded designs that will be a part of everyday components. Such environments will have new design difficulties that will have to be overcome as they become more prevalent in society [36]. Pervasive environments can range from smart homes that allow a user to remotely control the multiple systems in their home such as the heating, entertainment, and laundry systems; to a pervasively enabled jacket with a single system that controls the user's body temperature.

Pervasive environments are generally constructed by combining different systems that individually do different things. Thus, a smart home for example, will have separate systems for heating, communication, and entertainment, which work together to create the entire smart home environment [6]. Likewise a heating jacket would be composed of sensor systems and heating systems working in conjunction to fulfill the users needs [22]. As a result, pervasive computing techniques increasingly rely on incorporating many disjoint systems to create an environment that accomplishes more than the sum of its pieces.

The problem with creating environments from distinct systems is that each system is not built to handle the nuances of each of the other systems. Take the example of a smart home environment, one could imagine that in your home if your fire alarm went off it would send your cellphone a message alerting you, it may signal your sprinklers to turn on, and may signal all of your appliances to power off. This would be a fairly comprehensive fire prevention plan that would effect multiple systems including your cell phone, sprinklers, and every appliance in your house. Common fire alarms now have a feature that they beep whenever the batteries are low. In a pervasive environment each of the systems dependent of the fire alarm may not have been built to handle the difference between a fire alarm beep and an actual fire alarm,

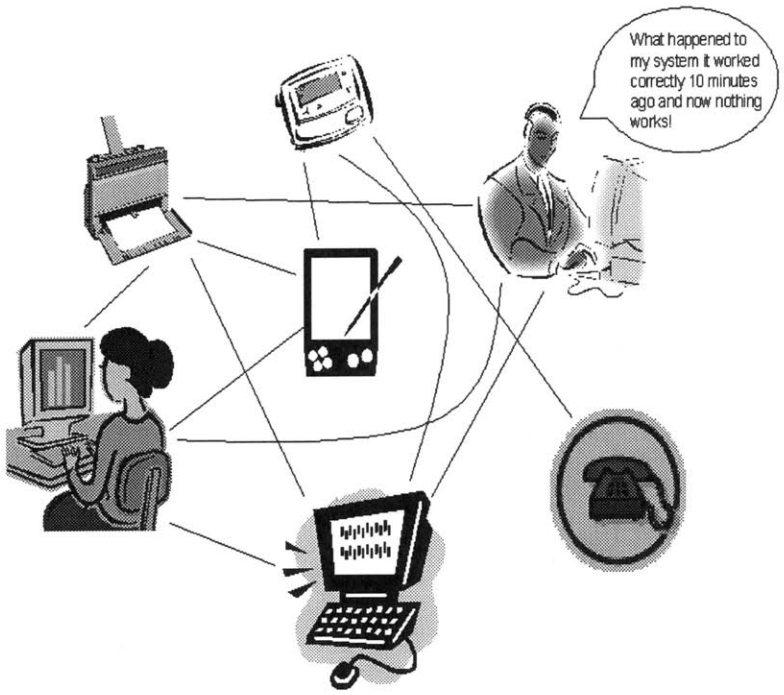


Figure 1-1: In a typical pervasive computer environment, a collection of devices, applications, and even users work together. When there is a faulty device or system, how can a user identify the source problem?

thus if a user's fire alarm beeped to tell them to get a new battery, they could instead end up getting a periodic cell phone page, their appliances would stop working, and even their home sprinklers could may come on repeatedly throughout the day[26].

This situation is problematic not only because it could happen, but one could imagine being totally dismayed when entering a home where the appliances don't work, the sprinklers are turning on and off, and not being able to rectify it quickly [30]. Without a system for obtaining knowledge about the environment, the debugging process would be arduous. A simple solution to this problem that is commonly used is to *reboot* the environment, however, one many environments have no mechanism to *reboot* an entire house, while ensuring that the systems start at the correct time in the correct order. The ideas presented in this thesis can play an important role in pervasive environments that are created out of disparate systems not created to handle the individual nuances of every other possible system they could interface with.

For the context of this thesis, *systems* will refer to the component systems of a pervasive environment and *environment* refers to the collection of systems working together.

1.2 Problem Description

Attempting to create a pervasive environment from various distinct systems not created to work together introduces many new problems into system design. These problems arise from a number of different causes including:

1. Dependent systems may not know how to react when a system they depend on fails or goes offline.

2. A system may start sending unexpected data (i.e. error messages or invalid outputs), but may still be operating according to specification, and other dependent systems may not know how to react to the unexpected data they receive.

3. A system may be in a corrupt state and be sending out incorrect information

4. Links could be added or removed between processes that create unexpected dependencies in the environment.

5. Changes could be made to one system that propagate to other systems, or problems that occur with interactions between the sub-systems of a system could cause problems in the overall functionality of a pervasive environment.

6. Race conditions may arise between different systems that cause an inconsistent state for the environment.

The main issue with these types of problems is not that the systems are acting incorrectly, but rather that a system is not acting as the other systems in its current environment would expect it to act. Therefore, a system could be causing problems in a pervasive environment, but need not be failing itself. With a plethora of possible

problem causing situations in every environment, it is virtually impossible for an environment designer to anticipate and provide solutions to every problem that could occur. A solution is necessary to allow the user to understand the possible sources of a problem in their environment and provide direction for rectifying these problems.

1.3 Traditional Solutions

There are many traditional solutions for debugging environments. These traditional solutions can range from a simple restart of the entire environment [30] to analyzing log files of a system and understanding the structure of what happened while the system was running [20]. Some of these solutions, such as *rebooting*, are based on simply ignoring the problem and restarting the system in the hopes that the issue will not arise again. Other systems are based on the assumption that the system will need to be shut down or stopped to analyze the log files and understand what exactly transpired in the system before making the necessary changes.

While these methods proved effective thus far, in pervasive environments that run for long periods of time with many different systems, analyzing system performance logs can get time consuming and difficult to understand. Furthermore, rebooting a system may not be an option. Thus, there is a need for a new solution that would enable users to quickly ascertain the problems in their environment and move to rectify these in an efficient manner.

This thesis focuses on analysis of transmission streams, which may generate a great deal of data. There is a need to analyze this data quickly and efficiently, using many techniques common in the mathematics and database analysis field. A few of these techniques have been in use for many years, yet are new within the field of pervasive computing. Database analysis techniques are particularly useful because database analysis often involves looking over a great deal of information a few times, many times only once, and attempting to deduce some relevant information from the data [37]. These techniques will be discussed in the course of this thesis.

1.4 Assumptions About the Environment

This thesis seeks to identify methods by which systems in a pervasive environment can be easily analyzed to facilitate debugging problems that may occur in the environment. More specifically this work focuses on exploring metrics by which systems in an environment can be ranked based on the largest change in their behavior, or health. Each of these health metrics will be analyzed with respect to a set of assumptions that are made about the structure of the pervasive environment and the resources available to calculate and retain these metrics.

The first assumption that will be made about pervasive environments is that they will be composed of a collection of systems. Many of the systems that compose a pervasive environment will not have been built to work with each other. Thus, these systems will have to be adapted to work together while still working according to their specifications [22]. Creating an environment that is essentially a system of systems introduces problems in that each system cannot be expected to handle each possible characteristic or output of every other system in the environment. At best, the systems will be designed to handle a majority of situations that the other systems may have. However, the borderline cases that occur will not be handled and can cause serious problems in pervasive environments.

Furthermore, creating an environment out of a system of systems introduces various communication inconsistencies. There is currently no predominant communication protocol in pervasive environments and the range of possible protocols includes TCP, Bluetooth, Firewire, IR, and Serial to name a few. With no standard protocol, it is difficult to create a single solution that would monitor the communication between these systems on all the different communication protocols and platforms. There are multiple reasons making monitoring systems on these low level protocols difficult, some platforms may restrict the protocols to certain users or access to the protocol on the platforms may be very difficult. Furthermore, with each protocol having different characteristics, there are different aspects to be monitored for each protocol. This not only makes monitoring difficult, but it also increases the complex-

ity in comparing the protocols. Given the various problems with creating a system of systems, it is important to understand how each of the metrics, for measuring transmission, that will be discussed can operate independently of the communication protocol and environment topology that is used.

The next assumption that will be made about pervasive environments pertains to the existence of a centralized server or control point. Some environments have a central point through which all communication occurs, while others may have a central control point from which to modify the environment and the systems within the environment. It is increasingly popular to develop environments that do not have a central focal point and are essentially distributed systems with autonomous nodes [35]. The advantage of a distributed environment is that there is no central server that needs to stay running correctly to support the environment. Additionally, each node can make decisions on what to do given a situation without having to wait for a centralized server to tell it what to do or affirm what it wants to do. This autonomy is important as systems can also be added or removed from an environment without having to make changes to any central server or control point.

With no central control point two distinct issues arise when considering pervasive environments. Primarily, it is extremely hard to monitor an environment if there is no central place that can be used as a monitoring point. As such, the metrics must be designed so that they need not be run on a single entity within the pervasive environment. Additionally, with no central location, the idea of *rebooting* a system is almost impossible as there is no way to reboot or reset all the systems in an environment at the same time or in a specified order by which they will operate correctly. It is necessary to build a series of metrics that enable a user to identify the problem systems within an environment so each of these can be reset or restarted independently as opposed to restarting the entire system.

The next few assumptions will be based on the systems themselves and what they can contribute to the overall monitoring and debugging effort. The first of these assumptions is that each system can monitor itself and provide information regarding a specific metric. This thesis will outline the metrics that will be used under the

assumption that the various systems will be able to compute these metrics easily, with minimal resources, and report the status of these metrics when asked. With no centralized monitoring utility, it will be the responsibility of the individual systems to keep track of their own behavior and report this metric when queried. It is assumed that this query will be contained in a simple API with a very small set of methods, and it will be the responsibility of the system designer to implement these methods.

One of the important advantages gained by having systems provide these metrics using their own implementation is that environment designers need not attempt to collect this data themselves. If designers were asked to collect this data it would entail a considerable amount of effort in modifying communication and software protocols such as the Bluetooth protocol or the java socket SDK. By placing the implementation on the system designer the environment designer need not worry about the implementation details or constantly having to create patches and updates whenever changes are made to the standard communication protocols.

The final assumption that will be made about the systems in an environment is that when computing the metrics, the systems will have a low overhead of processing time and storage space to work with. With a small amount of processor time, ideally logarithmic with respect to the output of the system, devoted to computing the system behavior metric, the metrics must be simple to compute and easy to monitor. Additionally, with a small amount of storage space, again logarithmic with respect to the system output, the systems will not have much space to store data related to the metric calculations.

Finally, the systems must calculate these metrics in real time. This will ensure that users are getting updated and current information about the state of their environment while the environment is still online and functioning. Furthermore, it will ensure that users need not wait to examine the functionality of their system by these metrics as they will be constantly calculated and updated. Each of these assumptions have been designed to minimize the effect of tracking system behavior, but at the same time allow the metrics to maximize the knowledge gained from a pervasive environment.

1.5 Goal

This thesis has specifically focused on metrics to identify the largest change in a system's behavior, as this is a useful characteristic to understand how characteristically or uncharacteristically a system may be performing. Systems that exhibit large changes from their normal behavior are more likely to be acting uncharacteristically. Subsequently, in attempting to identify problem points within an environment, it is helpful to look at the systems that have deviated from their behavior the most and start by exploring how these systems are performing individually. Using such metrics one could then rank systems in an environment by which had the largest change in their behavior. A user could then take this data and have a clearly defined path by which to start debugging the environment.

The metrics that will be explored in this thesis are based on a few basic assumptions, which will be described in depth in later chapters:

1. The environment will be composed of a system of systems.
2. The environment will not necessarily have a central focal point from which all communication passes or from which the environment control is run.
3. Each component will be able to provide a specified metric of its relative behavior.
4. Each component will have a limited amount of processor and storage capabilities to track the metric.

The metrics that will be explored in the context of these assumptions will be thoroughly discussed in the course of this thesis and center around transmission rate analysis and transmission data analysis between the systems in an environment.

1.6 User Interaction

To develop a solution that will enable users to find and rectify problems in a system, there are multiple scenarios and sub-scenarios that a solution should satisfy. The

following is a small list and explanation of the scenarios such a system would need to provide discovery mechanisms for:

1. The Data that a System Sends Changes

   (a) The system could have gone offline: For this to happen the system could have shut off or hung, or the communication mechanism between itself and its dependent systems could have become interrupted.

   (b) The system could be sending out different data: The system could have changed the data it was sending or it could have started sending invalid data. Invalid data refers to data that a system would not typically send out, such as error messages, data caused by failure, or valid data that another system does not expect.

   (c) The system configuration could have changed: Many times a system undergoes specific changes as a result of its environment. These changes can be a result of changes in other systems or simply valid changes made by the user that other systems do not know what to do with.

2. The Environment Configuration Changes

   (a) The environment throughput could have changed: A system could have gone into a sleep mode, or the communication network could have become congested, all of which would cause problems in the workings of the overall system.

   (b) Links between systems could be modified: Many times in an environment links between systems are modified, by either adding or removing links between two systems or modifying an already existing link.

   (c) An environment error could occur: In centrally controlled pervasive environments errors in a central or controlling server can manifest in many unique ways to the environment. Thus it becomes important to observe the effects that changes in a server have on the environment.

1.7 Outline

This thesis will spend chapter 2 discussing the context of prior and current work in pervasive computing analysis, and will also discuss some of the current analysis techniques from which some of the metrics were built. Chapter 3 will go on to an analysis of the transmission rate and transmission data metrics and will discuss the implementation and results of the metrics with respect to the aforementioned assumptions. In chapter 4 we will evaluate the metrics in terms of system interaction and will also propose methods by which to keep a higher degree of autonomy and longer degree of analysis for pervasive environments. The thesis will conclude in chapter 5 which will include a description of areas for future work.

Chapter 2

Related Work and Background

This chapter presents the assumptions made about pervasive environments and the systems that compose them. This discussion is useful in structuring the analysis of the various metrics that were used. Furthermore, a description of current pervasive environments and their design is given followed by some background into related analysis work that will be used in implementing the metrics used in thesis.

2.1 Middleware Pervasive Environments

To further understand the scope and requirements of these analytical metrics it helps to understand the pervasive environments and infrastructures currently in development. There are a large number of pervasive environments that are in development [10, 15, 32, 1], we will focus on a few representative samples for this background discussion.

A number of software based pervasive computing infrastructures, have been developed among these is the agent-based approach to structuring pervasive environments [17]. The agent model is focused on allowing many different software systems, in the same environment, to locate each other and communicate over a common medium. The environment isolates different systems as individual agents that can perform specific function. For example, a projector is a resource to display an image to a group,

and a speaker is a resource to project sound. Using these various agents, commands can be developed which specify a set of agents that work together to perform a task. When one wants to give a presentation, the controlling agent will turn on the correct software and alert the devices it deems necessary, such as a projector and set of speakers. The projector will project the computer's output, and the speakers will project the sound coming from the computer's speakers. To implement this idea a few systems have been implemented among these are Rascal [14], a high level resource management system that allocates specific agents for use based on a command, as well as Metaglue [7], an extension of the Java programming language, that allows systems to communicate with each other within this agent based framework.

Another pervasive environment, the Gaia Operating System [29], extends the traditional operating system model to physical spaces. Similar to the way an operating system manages resources on a computer, in the Gaia environment, applications share a common kernel, file system, and event manager. Individual pervasive applications then integrate with the Gaia OS when they enter the physical space controlled by Gaia, and use the Gaia OS infrastructure as the platform on which to run. These applications will use the provided infrastructure to perform their tasks as if they were part of an operating system distributed among different systems. All of the resource allocation, load balancing, and communication is then structured using techniques similar to the way an operating system performs these tasks.

Another approach to infrastructure in pervasive computing environments is a goal-oriented mechanism [27], by which a user specified goal is defined and the environment then satisfies this goal. The environment is structured as a set of *pebbles*, each of which can perform a specific function. The pebble system involves creating a wrapper around different processes that conforms to a standard API. The goal oriented nature of the system then enables the environment to allocate pebbles to a user based on their goals at that time. Since each pebble has a standard API, the communication is standard among all applications in the system. This defers from the agent based approach where each agent has a specific method of communicating with the other agents in the system.

These are a few of the many pervasive infrastructures currently being developed. Each of these infrastructures is able to give a solution to resource allocation, communication, and functionality within a pervasive environment. Many of the assumptions that were discussed earlier apply to these systems and thus these systems provide solid groundwork from which to analyze the metrics used to determine pervasive environment behavior. A cohesive solution that can be applied universally to each pervasive environment is necessary, and will be fleshed out during the course of this thesis.

2.2 Related Analysis Work

The metrics explored in this thesis are based on a set of general analytical techniques, some of which are used in other fields such as mathematics or database management. There are a large number of performance debugging tools that each have their own merits [9, 34]. This section will first cover some of the work that is being done to analyze current distributed environments, and will then move on to exploring some of the general analytical techniques that will be explored in the course of this thesis. The internet can also be considered similar to a pervasive environment in that individual nodes do not have any control over the entire environment. As such many of these current analysis projects will center on the internet as the source of a pervasive environment.

2.2.1 Previous Environment Analysis Work

There has been some work done in analyzing and monitoring pervasive environments, much of this work has centered around distributed computing environments. An ongoing project at Stanford University, the Pinpoint project has focused on determining problems in large, dynamic internet services [25] . This system involves tagging and monitoring client requests as they pass through an environment and then applying data mining techniques to the logs of these requests so that the failures and successes can be correlated with the performance of the system.

This system is beneficial in the sense that by tagging the information as it passes through the environment the system can gain relevant information about the environment and analyze this information to get an idea of what systems are failing and where problems may lie. The problem with this system is that an environment with such a monitoring implementation must be built with such a system in mind. In the case of Pinpoint, the authors modified the *J2EE* platform to incorporate their monitoring tools and then build the system using this modified platform. By essentially instrumenting the applications themselves to work in the monitored environment, the system designers are able to accurately understand what is happening in a pervasive environment.

Another environment was designed to debug a system of black boxes [24], and instead of instrumenting the systems themselves, the environment designers choose to instrument the environment within which the systems would be run. In this environment, messages passed between the environment are traced, similar to the Pinpoint environment. However, this environment seeks to discover the causal path patterns within the environment and give the user information about the latency within a system and the topology of the system. This system does a beneficial job of monitoring the transmission data that passes between systems, and does an especially good job of treating the systems as black boxes, so that they need not be built using a specific platform or toolset.

This implementation is an advantage over the previous Pinpont environment in that it treats each system as a black box. Unfortunately, this environment model doesn't perform the functions to be discussed, that give an indication about the performance of the environment. Simply observing the causal relationships in the environment does not give an indication of how the systems are performing or even how the environment as a whole is functioning. The system described in this thesis will address these issues and will not only treat the systems as black boxes, but will move on to discussing other relevant metrics that would be useful in analyzing the environment.

Researchers the University of Western Ontario have also done research on making

distributed applications manageable through instrumentation [21]. Their motivation was to provide a central a coordinated view of the entire environment, this research is relevant as it shows a need to monitor the status of a system and provide a monitoring service that can give an indication of the status of the system. This system however, depends on the existence of a central coordinating server that can monitor and control the environment, this idea is contrary to one of the first assumptions that has been made about pervasive environments. This thesis will discuss a plan to monitor pervasive environments that is not dependent on a central, controlling system.

2.2.2 Analytical Techniques

The metrics explored will be based on an analysis of the data that passes between the different systems of the environment and the characteristics of this data. In many of the systems previously discussed and especially in the system implemented for this thesis, there is a large amount of data that is to be monitored and understood, as a result it is much more efficient to do the analysis as the data is produced, so that one always has a current understanding of the system at that time. To analyze the data, various techniques will be used, including data stream analysis and clustering techniques. This section will review some of these techniques and they will be described in depth later in this thesis. Some implementation work has already been done in this area and includes building off of probabilistic counting mechanisms as described by Flajolet and Martin [13], data stream calculations described by Manku and Motwani [23], and counting distinct elements in a data stream as described by Bar-Yossef [3]. Most of these applications have been explored in the database management space where time and space resources do not come close to the size of the data being analyzed. These techniques offer a mechanism for maintaining statistics of data with minimal space and time lag, since space and time efficiency is an important feature in reducing overhead to individual systems.

There has been other work done specifically on analytical techniques that can be used in pervasive and distributed systems. Graham Cormode designed a system that

dynamically monitors the most frequent items in a system and keeps records of the performance of these items [8]. The advantage of this system is that it keeps track of these items dynamically and has been designed for a one pass view of the data that is being analyzed. Rebecca Isaacs also discussed some of the benefits of performance analysis in distributed systems, and spoke about the benefits of doing this from a data driven approach. Much of the same concepts descried by Isaacs will be discussed in the context of this thesis [19], including monitoring throughput versus response time and the discussion about offline processing of data.

Chapter 3

Transmission Analysis

This chapter will discuss in depth the metrics that were explored for monitoring system behavior. There were two main areas explored: transmission rate analysis and transmission data analysis. This chapter will describe these metrics, discuss how they meet the previously stated assumptions, explore their implementations, and how they performed.

By monitoring transmission data in a system there are a few characteristics of the data that is of importance. One of the most relevant things is understanding what exactly is being transferred while the other is understanding how well this transferring is happening. Each of these areas offer numerous possibilities for monitoring what is happening in a pervasive environment. This thesis has focused on specific means for gathering this data in ways that maximize the benefit while ensuring cheap operation costs.

3.1 Transmission Rate Analysis

The first metric is transmission rate analysis, which involves analyzing the rate of transmission of data between systems in an environment. There are many unique factors about an environment which can be discovered from this analysis, such as the active, static, or idle status of a system; or the causal relationships in a system.

These advantages will be discussed in the next section.

There are many advantages of having information about transmission rate analysis. By monitoring the communication in an environment, one can gain an idea of what devices are communicating with each other and more importantly how often this communication occurs. With this communication information one can then passively observe what systems may have gone offline, or what the environment configuration is. Monitoring these environment characteristics are integral in understanding changes that have happened within the environment.

An important characteristic that transmission data gives is the ability to understand the system state. There are various states that a system or component of a system can be in: active, listening, sleeping, offline, to name a few. Many communication systems that have multiple states will transmit a certain amount of information based on their state, thus, by tracking a system's transmission data, one can monitor the state of a system. For example, if a system is inactive it will output less information than an active system would output. Through this information gathering, transmission rate analysis can give rich information regarding the status of a system, the status of an environment, and additionally the topology of an environment.

Given the assumptions about pervasive environments, transmission rate analysis comprehensively meets each of these requirements. In recording the rate of data transmission between systems, the environment is treated as a collection of disparate systems. There is no need for a centralized server to monitor transmission rates as each system can monitor its own transmission rates. Furthermore, each system can easily keep track of the rate of data it sends and receives from other systems in the environment using a simple algorithm that will be discussed later. Finally, it will be shown that this algorithm requires a constant amount of storage and processing time on the order of the localized range in which transmission data is needed. As such, analyzing the transmission rate between systems provides rich information and can be accomplished within the scope of the stated assumptions.

The two techniques were used in analyzing the communication between a VNC environment [11]. A VNC application allows users the ability to remotely view and

manipulate the desktop of a remote computer. In this environment a communication channel is setup through TCP/IP allowing a user with the correct password to the vnc server to connect to this server using a vnc client running on a remote machine. The servers and clients are built for many different operating systems making it easy to use on virtually any computer.

In this environment a VNC server and a VNC client [2] were run on sperate computers and connected as nodes to a CORE [5], a common oriented routing environment, which essentially allows a user to route TCP/IP socket connections between two nodes that are ambivalent of each other's network connection. The advantage of doing this is so that each node need not be hardcoded to know exactly where all the other nodes it is connecting to are located. By setting up a core to listen and send on a common, established port it allows different devices to connect to this port and send information to the core. Nodes simply connect to a predefined socket on a core server and then communication can be routed to and from the node. There also exists a control connection to a core that provides configuration information to the core. Using this control connection one can route information from one node connected to the core to another node connected to the core. Thus, one node that is setup to send information to a core and another node that is setup to receive information from a core can communicate with each other.

For this implementation a core was setup on a single server. A vnc server was setup on another computer and was connected to the listening port of the core. Thus, the vnc server was constantly sending the core screen shots of the server's desktop, under the assumption that the core was a vnc client. Additionally, the vnc server was listening for the core to return mouse and keyboard events to send to the vnc server's desktop environment. A vnc client was then setup on another computer and was connected to the listening port of the core. The client was listening to receive screenshots from what it believed was the vnc server, but what really was the core. The vnc client was also ready to send mouse and keyboard events to the core whenever they were activated on the vnc client's computer.

This architecture was chosen for a few important reasons. The vnc environment

provides a rich, real-time mechanism that involves a network connection. Additionally, the setup of the core in the middle of the vnc connection is a bit counter-intuitive, given the stated assumption that the metrics would be designed so that no central server was needed. However, there are two motivations behind choosing a centralized core setup. The primary reason was that it allows for use of vnc which is a common and widely used system and an application that is common within the scope of pervasive computing. Secondly, since this is a testing implementation, it was desired that an easy mechanism be built to quickly modify and change the analytical metrics and algorithms used.

Had actual changes to the vnc software been made, it would have been a time consuming task to constantly rebuild the client and server whenever a small tweak was necessary to a given algorithm. After sufficient testing was completed with the algorithm, a final algorithm could then be implemented into the vnc environment for use as implied in this thesis. Additionally, at the onset of this research it was not clear what type of API would be necessary to view the metric data, thus, by simply modifying one core server a useful API could be decided upon at leisure.

Overall this vnc and core environment is a small but interesting pervasive environment to analyze various things that could happen: the client or server could crash; the link between the two could break; or either side could go idle. Tests exhausting all of these properties were performed and very interesting results were gathered.

For this implementation the core essentially acted as a wrapper around the individual nodes, in this case the vnc client and vnc server. All the analytical metrics were implemented within the core and an API was built within the core to access the necessary data from the metric calculations.

3.2 Transmission Rate Technique and Results

There is a trivial method for calculating the rate of data transmission between two systems. This involves periodically counting the total amount of data passing through a communication channel and then dividing this by the total time over which the data

was gathered. This implementation is sufficient for an environment that does not have volatile communication speeds. Unfortunately, pervasive environments do not enjoy this luxury, often systems will vary drastically in the amount of data they transfer, depending on what they contribute to the overall environment. As such, a system that averages the data transmitted over the entire duration of monitoring is often irrelevant when attempting to indicate what a system is transmitting at a given time.

An improvement over the trivial data transmission calculation is one where data transmission rates are calculated for a given window of time. For example, the amount of data transmitted every 10 seconds can be calculated and then stored. This information proves useful as one can then use these small data windows to get an approximation of how the transmission is proceeding. If the data starts getting smaller one knows that the transmission is slowing down, and so on.

There is a downside to this windowed method of analysis. While one is given a better idea of what is happening in the system at the current moment, it still requires additional work to figure out what each of these small transmission rate windows means within the scope of the entire system performance. Furthermore, many communication protocols can be *bursty* over short periods of time, making it hard to accurately gauge the properties of the transmission rate, using a small time window. Unlike the trivial calculation, where there was too much forgiveness given to volatile transmission rates, the windowed scenario is not forgiving enough for volatile transmission rates.

To add some robustness to the analysis, it is helpful to look at the standard deviation of the windowed transmission rates for a larger window of time. For example, if there are windowed transmission rates taken every 10 seconds, then it is helpful to look at the standard deviation for a given set of these windows (i.e. 6 or 12 windows, or 1 to 2 minutes) to get an idea of what the transmission properties have been like for a larger period of time. This windowed standard deviation clears up much of the volatility in the transmission rates, but also gives a clear indication of what trends there are in the transmission rates.

An algorithm for this is analysis is fairly straight forward, minimizes space con-

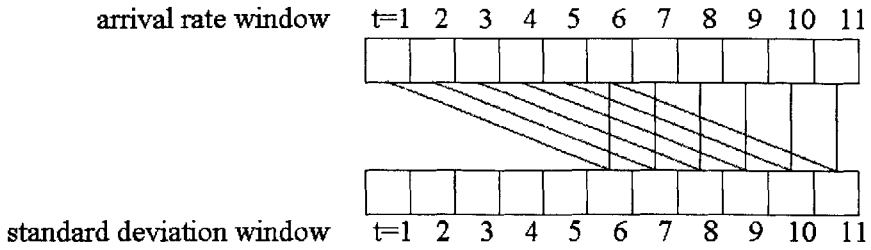


Figure 3-1: The transmission rate analysis calculates a *windowed* standard deviation over the past 6 arrival rate windows. Each of the boxes on the arrival rate signify an arrival rate window, the standard deviation of the past 6 windows create the entry for the standard deviation window at that time.

straints, and can be accomplished in constant time. For this algorithm we will assume that it is trivially possible to monitor the amount of data transmitted over a communication channel between two systems in an environment for a small window of time. The first step is to create a circular buffer, the size of the number of windows the standard deviation is to be calculated over. For example, if one is hoping to calculate the standard deviation at a given point for the two minutes prior to that point and a windowed transmission rates of 10 seconds was used, the vector would be of size 12. As each 10 second window of transmission data is recorded, it would be stored in the oldest place in the buffer. This ensures that the buffer contains only the transmission data relevant at a given time. To take the standard deviation of this data, a trivial standard deviation algorithm can be used to calculate the standard deviation of all the data within the current buffer. The standard deviation could also be taken every time window and stored in a separate array.

After connecting the vnc server and client to a core, outfitted with the correct metrics, various tests were performed implementing the described algorithms for transmission rate analysis. The average transmitted bits per second was recorded for 10 second intervals by both the client and the server. As mentioned this data was calculated within the core and was then accessed using a standard set of API's. In this case the API consisted of two simple methods *getRate()* which returned the average arrival rate of data for an interval of ten seconds and another method *getStdDevRate()* which

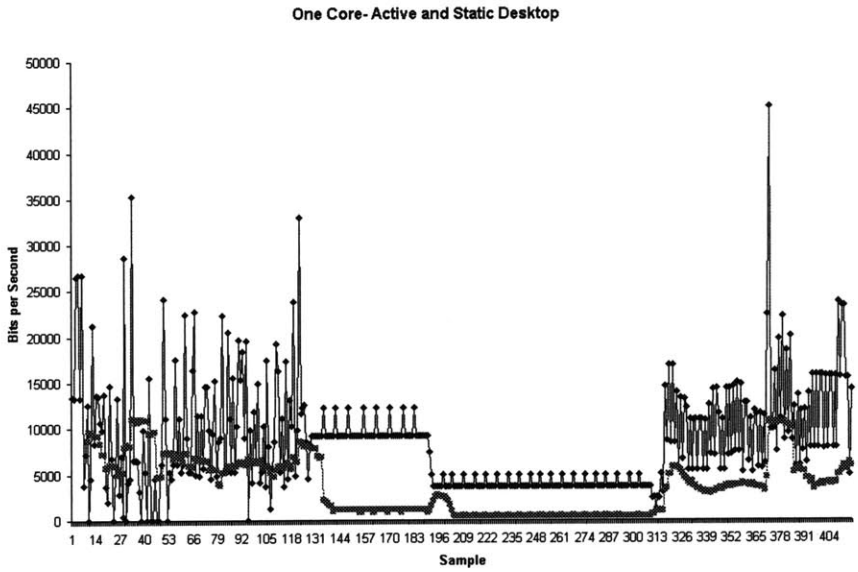


Figure 3-2: Transmission rate data for a normal run of the vnc application. The darker line indicate the arrival rate at the time window while the lighter line is the standard deviation of the past 10 time windows.

returned the standard deviation of the arrival rates within the vector of the past 10 ten second window sizes. These window size and standard deviations were recorded by another computer that queried the client and server portions of the core for the transmission rate information and kept a record the information that was reported. The data that was found and stored by this other computer has been graphed to show a few notable results.

The transmission rate data that was found proved highly valuable in evaluating the state of a system, there are a few plots of data that will be presented to show the variances in the data transmission rates.

The first plot is the transmission rate for the vnc application described above. The vnc server application operates by sending the differences in screen shots of the server to the client, thus when a large amount of activity happens between the client and server desktop environment (i.e. the client opens and closes many files on the desktop of the server), the transmission rate rises, and when there is less activity the transmission rate falls. Figure 3-2 shows a standard run of the vnc application where

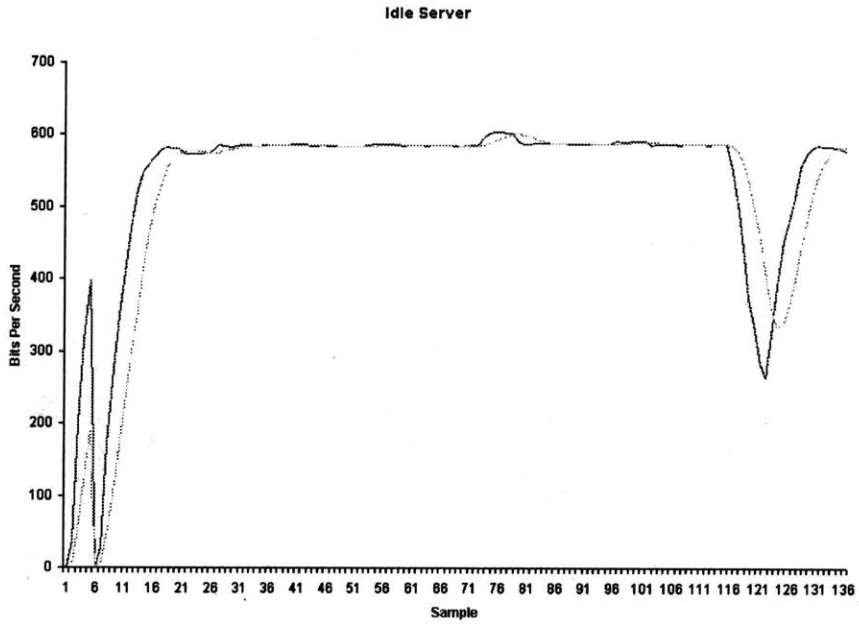


Figure 3-3: A graph of the average bits per second for the vnc transmission rate analysis implementation. The drop in data rate denote the time that the server became ide, the subsequent increase was when the server was made active again.

a client connects to the server and has an active and static section. The application had active and static periods, these were created by having the client open and close a lot of programs on the desktop, then letting the desktop go idle for a while, and then activating the desktop again. As can be seen in the graph, the data is fairly active in the beginning, but then dips when the client stops using the desktop, only to rise again when the client reactivates the desktop. For this and all subsequent figures in the next two chapters the two lines denote the two data points tracked. The darker line is the arrival rate of the data over a ten second window, and the lighter line is the smoothened standard deviation of this curve found using the *getStdDevRate()* method.

To test out further features of the transmission rate analysis metric, a few other tests were performed to gauge these results. The first of these tests was to see what the effect of idling the server would be on the transmission data. When a computer is not in use the computer can idle some of its components, including the monitor

to save energy. As figure 3-3 shows when the server went idle and the screen saver started, less data was transmitted from the vnc server to the vnc client, which can be seen by the dip in the two plots. Furthermore, when the server was made active again, the transmission rates rose.

This idle test proved valuable because it showed that a difference could be detected if a component of a pervasive system went idle but was still functioning correctly. When a server goes idle and a screen saver is activated, less components are running, however the server is still functioning correctly and would respond positively to *keepalive* or any other probes to test uptime. This transmission data analysis would therefore prove useful if attempting to observe the active or idle state of a server.

Another observation that was interesting was to monitor the arrival and outgoing rate of data through the environment. In this case the outgoing rate of data from the vnc server was observed along with the arrival rate of data to the vnc client. Since data passes directly between the two systems, one would assume that the curves would look similar. This concept can be seen in figures 3-4 and 3-5 which plot the outgoing rate of data from the vnc server and arrival rate of data to the vnc client respectively. There is a strong correlation between the two graphs that can clearly be seen.

There is indeed a correlation between data that is sent from a vnc server and data that arrives at a vnc client. This concept could prove to be useful in observing many other circumstances that plague pervasive environments. In other tests that were run in a similar environment, when the communication link was broken between the vnc client and vnc server, the vnc client side transmission rate dropped to 0 since it had stopped receiving data, but the server side remained active as it was still sending out data expecting the client to receive it. When the vnc client crashed in another test, the vnc server side data reported similar characteristics, while the vnc client side did not respond to any queries, and thus no plot was visible. By not responding to queries the vnc client side clearly had issues, while in the previous example when the vnc client responded with an average of 0 bits received, it indicated that the vnc client was functioning and that a possible problem had occurred in the link between

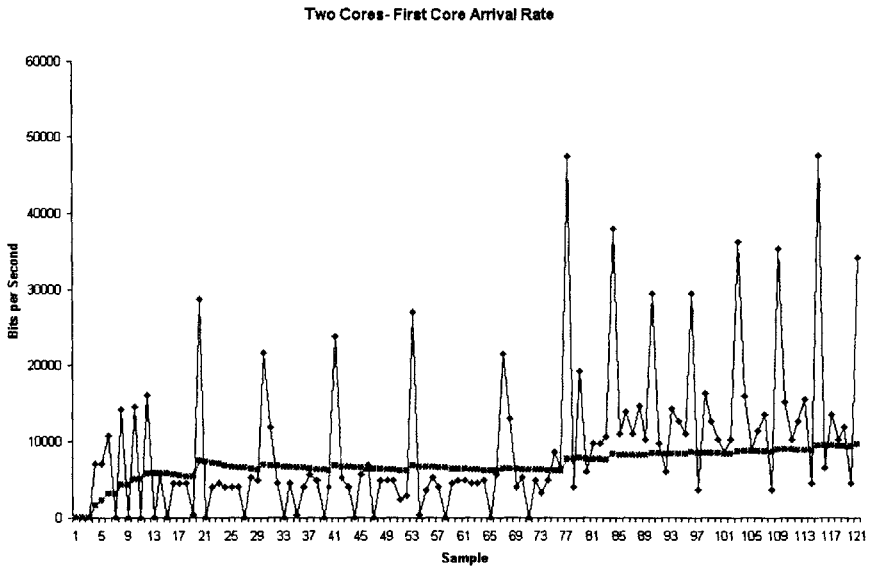


Figure 3-4: A graph of the outgoing rate of data from the vnc server to the vnc client in the vnc application. The darker line indicate the arrival rate at the time window while the lighter line is the standard deviation of the past 10 time windows.

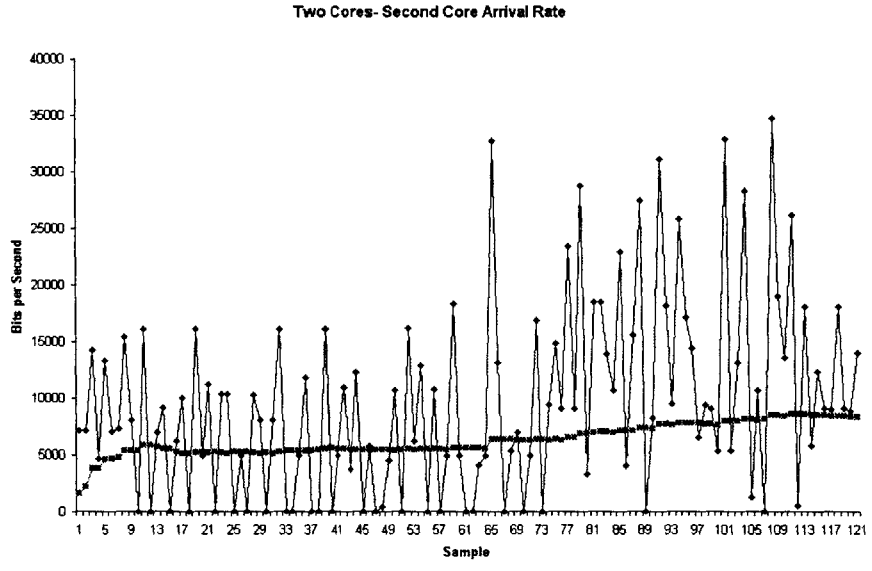


Figure 3-5: A graph of the arrival rate of data to the vnc client from the vnc server in the vnc application.The darker line indicate the arrival rate at the time window while the lighter line is the standard deviation of the past 10 time windows.

the vnc client and vnc server. When the vnc server crashed the vnc client reported receiving 0 bits but the vnc server did not report anything, indicating that it had crashed.

Building on this concept is the idea that in passive systems where the direction of communication is not known, one could observe the communication in a system and deduce which systems are talking to each other by correlating the communication between them. More on this subject will be discussed in the future work section of this thesis.

It has been shown that monitoring the transmission rate between systems in a pervasive environment lends to extensive information about the stability of individual systems, the structure of the environment, the performance of the systems, and also the performance of the environment as a whole. In this experiment, by simply having the client and server monitor the data they individually received and sent over a short period of time, a rich amount of information about the performance of the environment was gained.

3.3 Transmission Data Analysis

The second area that was explored was in analyzing the data that passed through a system. This metric was explored by examining the data that was sent and received by each system, and recording the number of distinct elements that had been seen in a given window of time. This analysis was able to yield some results about the environment performance and individual system properties.

Keeping track of the number of distinct elements that have been seen yields many clear advantages. The first of which is that by noting differences in the number of elements that pass through a communication channel, one gets a better idea of how the systems participating in that channel are functioning. Often, if a system hangs or fails, it will repeatedly send out the same data over and over. Thus, the number of distinct elements could drop from a larger number to 1, which would indicate that the system has failed. If one was just doing a transmission rate analysis of this

communication path, the transmission rate may not have changed, but by analyzing the actual data that passes over the system, one could get a better idea of how the system is performing.

Furthermore, when a system is performing reliably it may output the same amount of distinct elements over a period of time or process cycle. Thus, by monitoring the number of distinct elements one could have a reliable bound on the performance of a system. It is pertinent to note that this technique need not know the content or context of the data that passes through the communication channel, but rather just be listening to the bytes that passes through the environment.

Tracking the number of distinct elements transmitted in the environment gives information about the performance of a system, while also meeting the previously stated assumptions about the resources in an environment. Treating the environment as a system of systems and tracking the data that gets passed between the systems allows for the systems to be monitored as individual nodes in an environment and need not entail treating the environment as a cohesive system. Furthermore, the algorithms available to count distinct elements in a window operate with a low overhead and do not require any centralized server or monitoring point. Thus, the distinct element tracker meets all the assumptions that were made about pervasive environments.

3.4 Transmission Rate Technique and Results

The algorithm to count the number of distinct elements is fairly straightforward and was discussed fairly concisely by Flajolet and Martin [13]. This algorithm is based on the use of a universal hash function, a hash function that has an equal probability of hashing a given key to each of the possible values in the range, to minimize on hash collisions. Another feature is the *bitmap* vector which is the length of the binary representation of the length of the numbers in the domain of the hash function and will be used to approximate the solution.

The fundamental concept of this algorithm lies in the properties of the universal hash function and some basic probability. If one had a number of things and was

randomly assigning these things to a set of possible options, there is a logarithmic scale by which the distribution of things to options takes place. For example, if one had n balls and n buckets, when the balls were randomly assigned to the bucket, one would expect about 1/2 of the buckets to have at least one ball in them. If there were $2n$ balls one would expect three quarters of the buckets to be full and likewise if there were $n/2$ balls one would expect about one quarter of the buckets to be full.

Using this trend, if one didn't know how many balls they had, they could just throw all their balls into a set of buckets and based on the number of buckets that were filled, approximate the number of balls they had thrown. Taking this a step further, if one threw the same number of balls into various groups of buckets, each group with an increasing number of buckets. One could analyze each group and see which group had half the buckets full and then approximate that the number of buckets in this group was the total number of balls that they had. A further optimization that one could do is choose a bucket and look at this bucket in each of the groups. As one analyzed groups in an increasing fashion, the transition where the bucket had a ball in it and the next larger group that didn't have a ball in it, then the number of buckets in this smaller group is probably the number of balls the user had. Now that we have gone over a basic description of the background of the algorithm, we'll move on to a discussion of the actual algorithm and the implementation.

The algorithm begins by taking each transmitted piece of data and hashing it to a binary number using a universal hash function. Next, take the position of the least-significant one bit and set the corresponding bit in the *bitmap* vector, if it has not already been set. The algorithm will continue to do this for the subsequent data that enters the communication channel. The expected value of the number of distinct elements that have passed through the communication channel since bitmap was reset is $\lg \phi n$, where n is the position of the most-significant (leftmost) 0 in the *bitmap* vector and ϕ=0.77351. By nature this algorithm doesn't yield an exact number, but rather yields an estimate of the number of elements that have passed. This estimate is within an order of magnitude of the correct value.

As was mentioned earlier it is important to have an idea of what elements have

arrived in a recent window of time. There is a simple solution to this problem, using the current algorithm. Instead of using a vector of binary numbers to store the *bitmap* vector, use a regular vector and keep a count of the number of elements that have been seen. Whenever a new element arrives, mark the position of its least significant 1 in the corresponding place in *bitmap* with the timestamp of the element as opposed to marking it with a 1. Thus, when considering the number of distinct elements that have passed in the most recent window of time, all one needs to do is find the value in *bitmap* that is less than the current time stamp minus the window size and use this value as n.

This analysis gives an idea of how many elements have appeared in a given window of time. The next step is to smooth out this curve as was done previously in the transmission rate analysis section. To do this the number of distinct elements that arrive is recorded in a vector which has a size of the number of windows that are wished to be recorded. Thus, if one wanted the standard deviation of the past 10 windows of distinct elements, this vector would have a size of 10 and again a trivial standard deviation algorithm could be used to calculate this standard deviation. Furthermore, the vector would have a counter that is constantly incremented whenever a new number of distinct elements was added with every window size. This number n would then be added at the location n mod 10, to ensure that the vector only held the 10 most recent distinct value calculations.

The implementation of the distinct element calculation was done in a similar way as the implementation for transmission rate analysis. A vnc application was used in conjunction with a core. The core acted as a wrapper around the vnc server and vnc client, and gave the ability to monitor and access the data that was being sought. A separate server monitored the core and queried it for information about the vnc server and vnc client transmission.

Once again there were two methods as part of the API for the transmission data analysis, *getNumber()* which returned the number of elements that had appeared in a 10 second interval, and *getStdDevNumber()* which returned a standard deviation of the number of elements that appeared in a given window of time.

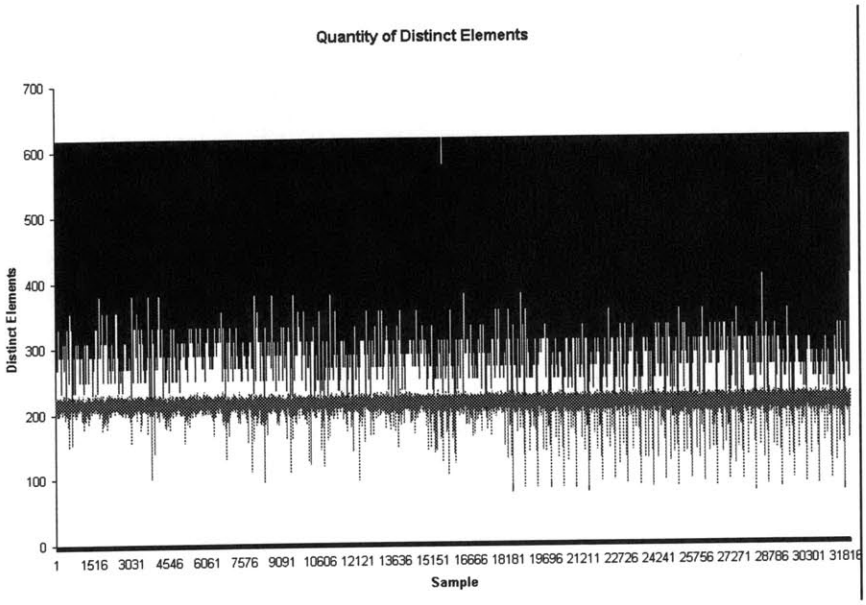


Figure 3-6: A graph of the results of the distinct element counter. The estimate always yields a number less than the actual number of distinct elements. In this graph the darker lines are the actual number of elements while the lighter graph is an estimate of this using the described algorithm.

The toughest part of the implementation for the distinct element counter was choosing a window size that would yield relevant information over which to count the number distinct elements. This exploration process took a bit of effort to pin down and after thorough testing was able to yield relevant results. The window size that was settled on was 10 seconds. Larger window sizes proved to have too little volatility to show relevant results in the vnc system and smaller window sizes appeared to have too much volatility.

Another issue that arose in the implementation of this algorithm was that the estimate was always within an order of magnitude lower than the correct value. As can be seen in the first graph of the transmission data results 3-6, the estimate is less than the data transmitted. However as can be seen by the graph the estimate still reflects the stability of the data. In this case, the data plotted is sent from a vnc server with an active desktop. As can be seen in the graph there was not much deviation in the amount of data that was sent out from the server and furthermore, the estimate of the number of distinct elements moved closely with respect to the

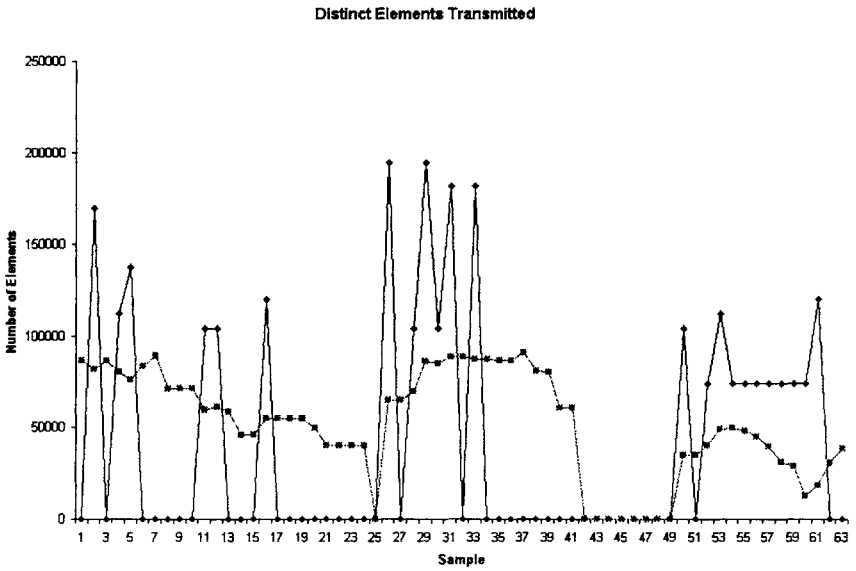


Figure 3-7: A graph of a normal run of the distinct element counter. The darker line is the estimate of the distinct elements sent while the lighter line is the standard deviation of this for a recent set of time windows.

actual number of distinct elements.

The reason that the estimate does not surpass the actual value comes from the process of storing the most significant 1 in each distinct element that is recorded. Since each element will probably actually be greater than 2^n (where n is the most significant 1), the actual value will be more than the estimate of the data. However the actual value will never be 2 times the estimate, as that would indicate that there was an even more significant 1 in its binary representation. The estimate yields an interesting and relatively accurate gauge of the number of distinct elements, however does not indicate the exact number.

The next figure 3-7 shows a normal run of the vnc server where it was both active and idle. As you can see, when the server went active, there was a significant drop in the number of elements transmitted. On the graph it appears as if there were no elements transmitted when the server was idle. This is a bit deceiving the number is in fact greater than 0, however, with respect to the other windows where 50-250K different elements were transferred, the number is effectively 0, since there were probably just a few different numbers that were transferred when the server was

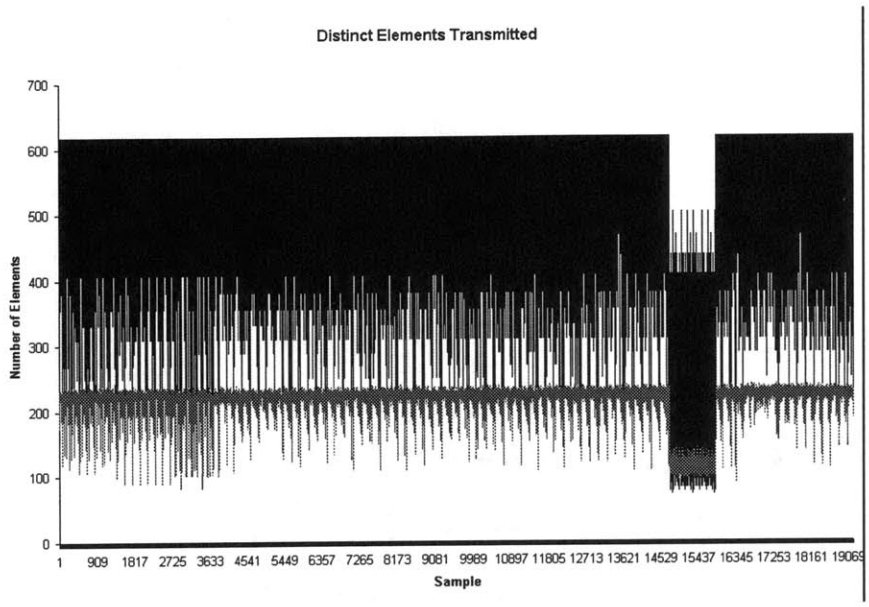


Figure 3-8: This is a graph of the vnc application were a single program dominated the screen of the vnc server for a period of time and only a few screen changes were sent to the vnc client.

idle.

In any case, the figure 3-7 with active and static aspects shows a considerable amount of information about the system. When the graph stabilizes it shows when a single application was dominating control of the desktop. Additionally, when the number of distinct elements drops to near 0, it indicates when the server has gone idle. It is worthwhile to note that this graph was created by running multiple different applications on the desktop throughout the active period of the test.

Another test was done to see what the effect of having one application that dominated the screen would have on the number of distinct elements that was transmitted. For this test the server and client were setup as before, but after a while a simple pinball game with a rich user interface was run on the desktop of the server. In this game the background of the pinball game dominated much of the screen and as a result the differences in the screen did not incorporate the entire screen.

As mentioned earlier the vnc server only sends the differences in the screen to the vnc client at every transmission. Therefore, only a small segment of the screen which reflected the differences in where the ball was currently located was transferred

to the vnc client. Given this rich user interface of this pinball application however, each difference was still a considerable amount of data. The timeperiod that the pinball system was activated can be seen in figure 3-8 by the dip in both the actual and estimated distinct element plots, where the actual number of distinct elements is lighter than the estimated number of distinct elements plot.

Another interesting test that was done was a variation of simply counting the number of distinct elements transmitted from the vnc client to the vnc server. In this experiment the size of each of the distinct elements was counted. This variation provided a mechanism by which to monitor the types of communication that was occurring over the connection. As can be seen in figure 3-9, whenever a large burst of data was transmitted, there was a clear difference in the size of the data that was transmitted. The two clear spikes in this graph were produced by streaming an audio file from the computer running the vnc server to an independent computer on the network.

In this experiment the low points of the graph indicate periods where just the vnc screen shots were sent from the server to the client. The large peaks in the graph indicate many interesting things. The initial burst was produced when the vnc server initially connected to the vnc client and transmitted the preference and initial setup information. The next two bursts happened when an audio file was streamed from the server to the independent computer.

This streamed file was completely independent of the vnc application, but provided some important information regarding the nature of the outgoing communication from the vnc server. In many pervasive environments different nodes will be used for different things and often more than one thing at the same time. Thus, it is important to track the traffic a server sends out and how characteristic this traffic is with respect to past performance. In this experiment it is clear that the server had many different forms of communication which independently caused the server to have different outgoing communication rates. Communication such as this, from a pervasive system, could affect the performance of the system or more importantly could affect the performance of systems that depend on this system for information.

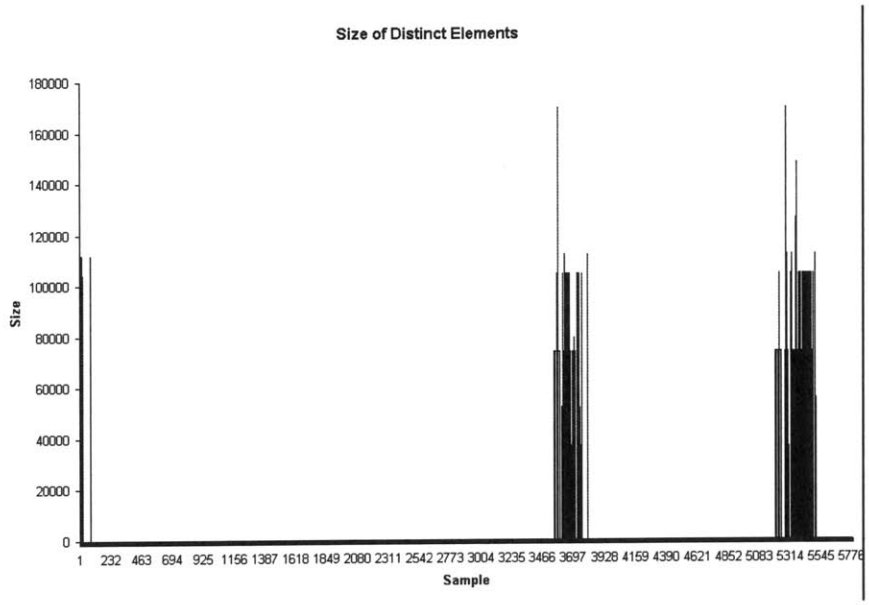


Figure 3-9: This is a graph of the vnc application that kept track of the size of elements that were sent from a vnc server. The spikes indicate times where there was a burst of data transmitted.

The past two chapters have provided a comprehensive look at the primary metrics that were observed and implemented: transmission rate analysis and transmission data analysis. As has been shown a good deal of information can be gleaned from a pervasive environment without having to violate any of the assumptions that were stated about the functioning and resources of a pervasive environments. Furthermore, the data that was gathered from this implementation lends itself to further work in other areas. The next chapter will discuss the merits of these methods with respect to pervasive environments and will move on to a discussion of other techniques that could be useful in gaining more information about the environment.

Chapter 4

System Design

This section will evaluate the applicability of the previously described metrics with respect to pervasive environments, and will then move on to a discussion regarding other methods of data aggregation and analysis that could prove useful when applied to pervasive environments. The section will conclude with a discussion about the overall effectiveness of the presented methods in the scope of pervasive environments.

4.1 Health Instrumentation

The two main techniques explored in the scope of this thesis were transmission rate and transmission data analysis. As was shown, both proved to be very highly useful in producing relevant and interesting information about a pervasive environment. While additionally minimizing the amount of processing and storage needed and providing a passive mechanism for calculating these metrics.

The passiveness of the system is the first positive that these metrics employ. Using these metrics an observer need not know what exact data is being sent from one system in an environment to another system in the environment. Additionally, each system is allowed to track whatever metrics it chooses, and report these metrics only when asked. Thus, a system need not be dependent on a central monitoring system or constantly report to a system that is always up and running. This is a

fairly liberating idea as it allows for more freedom and flexibility when designing and running a pervasive environment, since there are less dependencies to plan for and as a result fewer problems will occur.

Another feature that these metrics give is they provide the user with a richer understanding of the pervasive environment. Currently most systems do not have an interface for providing the user with the status of the systems within an environment or a status of the environment itself. The most a user understands is if a system in the environment is working or not working. Including these metrics in an environment will allow the user to have a better idea of what the individual systems are doing. If a system is idling, overly active, or no longer running, these metrics will allow a user to see the status and then be able to deduce problems in the system.

A further mechanism that these metrics provide for is change point detection. Change point detection is important because in large pervasive systems that run for long periods of time it becomes important to know when a system changed, to determine if a system is faulting or if it is responding to a change in another system. These metrics provide for a thorough change point detection system which enables the user to have a good idea of what the problems with the environment are.

In the example of the fire alarm that was provided in the introduction, a user would be able to enter the environment, examine what devices changed recently and then easily deduce that since the fire alarm was the first thing to change, the fire alarm is the primary target as initiator of the fault. This simple change point analysis would lead to a much better understanding of pervasive environments, and additionally, provides the user with a chronological list of changes that happened in a system from which to start debugging problems that may have occurred.

Furthermore, by allowing the user to have more information about the pervasive environment, environment designers need not worry as much about planning for every possible fault in a pervasive environment they build. Currently, since environment designers assume that the user will have very little knowledge of the environment, they must plan accordingly and have checks and fixes for every single possible fault in an environment. As environments get larger and more complex, it is almost impossible

to require an application designer to plan for every fault, mistake, or situation that may arise in the environment. Placing some of this burden on the user to understand and rectify a situation alleviates some of this burden from the environment designers scope.

The fact that these metrics can be provided in real-time is a very rich resource in a pervasive environment. As opposed to analyzing log files or understanding how a system faulted after the environment is shut down, environment users can now have real time understanding of the way the system is performing. This not only allows users to immediately fix any problem they may have, but it also provides users with an opportunity to rectify problems without having to shut down the entire environment or analyze the logs of what happened.

Since each system provides an API by which to access its metrics, these metrics can be utilized by any and all systems in the environment. This allows other systems in the environment to monitor a system and make adjustments accordingly. A situation where this may occur is that if two systems are communicating and another system wants to communicate with these systems, that system could monitor the traffic of the system it wanted to communicate with and only send its transmission when the target system stopped communicating.

Along with the upsides of these monitoring metrics comes other issues and problems with them that must be addressed. The primary issue is that in using such a technique the burden on monitoring an environment is placed solely on the user of the system. In other words, it is the responsibility of the *monitored* to understand what the system being monitored is doing. In some cases this is good, because it frees the individual system from interpreting its performance. Unfortunately, this places the burden of understanding what is happening in a system solely on the shoulders of the *monitorer*, the system or user that is doing the monitoring. The *monitorer* may not have the knowledge to understand what the system is doing and as such it may be problematic to interpret the metrics. For this reason it is important that the metrics chosen be inherently simple and easy to understand what changes may be occurring. Having information about the transmission rate and transmission data are

fairly simple ideas to understand which permit the *monitorer* to build from a simple base and deduce how the current situation leads to further understanding about the system.

The fact that each system must contribute a specific API is a fairly restricting idea. If each system must implement an API in a standard format for all other systems to consume, it places the burden on a system designer to understand which metrics to provide and how to implement these metrics. While the monitoring of these systems itself does not affect the system, providing this API is a change in the current mindset of system designers. Given the scope of this thesis, by shifting to providing this API, designers will be able to provide their users with a rich amount of information that they could use to understand the system itself irrespective of the environment that the system is in.

As with any system there are positives and negatives that characterize this system of calculating metrics. However, these metrics have been shown to give rich information about the environment and additionally, lead to a better understanding of the systems in an environment. The next section will move onto discussing methods to actually monitor these metrics that have been discussed.

4.2 Health Monitoring

Within the given requirements it is important to note that storage and access of the data is relevant to the performance of these techniques. To provide a real-time analysis of a pervasive environment, one does not have much opportunity to perform an in depth analysis of logged data or to create large storage databases. Many of the possible monitoring points will have limited storage and processor requirements. As such it is important to save on the necessary data and additionally provide some mechanism with which to access this data easily and in a consumable manner.

A method to monitor each of these systems in a comprehensive fashion is to have a separate system in the environment whose sole job is to query the other systems and analyze the data that is sent back. It would be the responsibility of this system

to interpret these metrics and provide some useful information from this.

Having a separate system as part of the pervasive environment is a powerful idea that has a good deal of merit with respect to pervasive environments. If a separate system is developed it can be its own independent entity and need not depend on other systems. Additionally, by not being an integral part of the environment, it can be free from things that effect the environment and provides a common place from where to gain information within a system.

This idea of having an health monitoring system that monitored the other systems in an environment was tested using a flash application as described by Hana Kim[16]. In this application many different systems are used including a speech interface, speech parser, and flash animation. In the application, a user can say things that trigger events in the flash animation. For the purpose of this thesis, a simple demo was designed by which each of the parts of the system monitored a few metrics such as the transmission rate and transmission data metrics that have been described. Then this system had a chat interface attached to it that allowed a user to send a chat message to the monitoring system and view how the system was performing. A screen shot of the system can be seen in figure 4-1.

This health monitoring system proved useful in understanding how parts of the flash application were performing and was especially useful as it provided a simple real-time method to evaluate the specified metrics.

Another advantage of having health monitoring analysis of the metrics is that it provides a storage point for the data reported by the systems. In each of the implementations presented in this thesis did not provide for any long term storage of data beyond a few time windows. Providing a longer term storage area allows for further analysis of the data over a period of time.

Another very important factor is the ease of access to this metric information. A separate monitoring server could serve this environment in many ways, including a way to provide user information in a consumable format. The flash animation implementation provided a chat based solution to delivering information. Other methods could be speech based or graphical user interface tools that provide the metrics in a

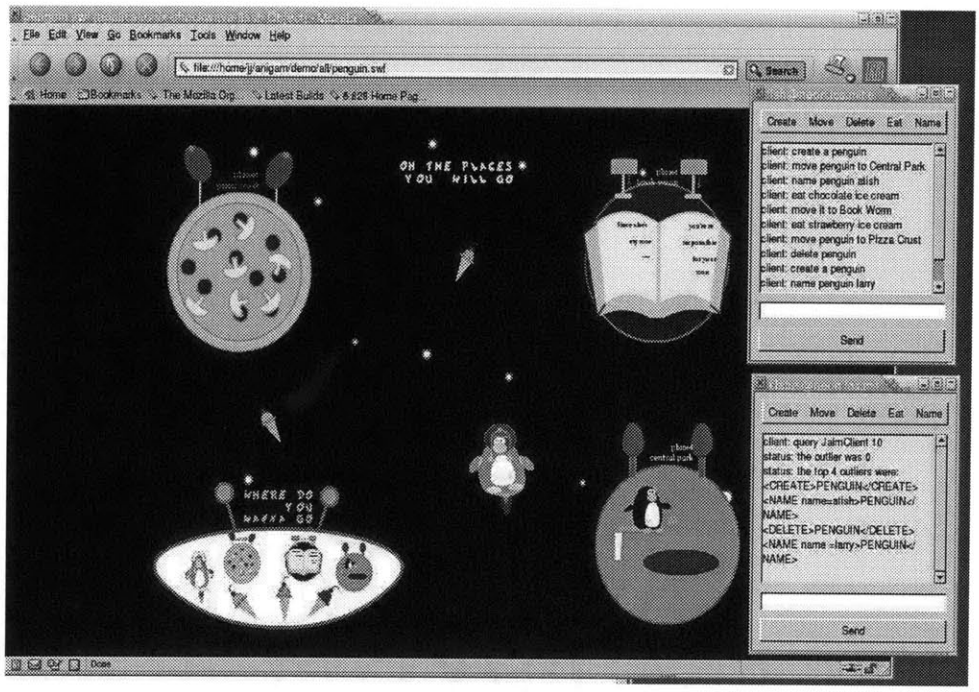


Figure 4-1: This is a screen shot of a speech enabled flash demo that allows users to send chat messages to an health monitoring monitoring server that provides information about the system.

consumable fashion where they can be interpreted by the user.

Finally, such a system devoted to storing and providing interpretation methods for these metrics could not only be independent from the environment, this system would be able to provide this data in real time to the user as that would be its only function. As each of the systems are providing real time data, this system can also simply aggregate the data and provide it in a consumable fashion.

There are many positives for an health monitoring storage system to keep track of the various metrics that have been previously discussed.

4.3 health monitoring System Design

An health monitoring monitoring system would provide numerous benefits to pervasive environment monitoring. This section will review a design plan for a pervasive environment and the features it could provide. This description can give environment designers a starting point from which to design a pervasive environment with

the previously described techniques.

The system will essentially be composed of two different aspects. The first of these aspects is the pervasive systems that compose the environment and the second aspect is the health monitoring monitoring system itself. As was mentioned earlier, each of the pervasive systems will have a process monitoring system that not only monitors the metrics to be tracked, but does so for each pervasive process on a system. Thus, if a pervasive system has multiple processes each doing different things, this monitor will observe each of these different processes and differentiate these processes within the environment.

Furthermore, the monitor will have a simple API which will return the results of the metric in question for multiple different time windows. These time windows will be of increasing magnitude and each of the metrics will be stored until it is superseded by another time window. For example, a system may have time windows of 10ms, 1s, 10s, and 100s, in this case the system would retain the 10 most recent 10ms window metrics, the 10 most recent 1s metrics and so on. Thus, the *monitorer* will be able to determine the level of monitoring granularity required by the pervasive environment and look at only those metrics of relevance. Additionally, as mentioned earlier, the burden will be on the *monitorer* to query and process this metric data, the system itself will only need to provide the metric and have a simple API for accessing these metrics.

The health monitoring monitoring system has the responsibility of querying the various systems for their metrics and processing this data into a consumable format for the user. Within this space there are many possible variations that are yet to be resolved. The primary variation is the frequency of which the data should be queried. This frequency is important because it determines the rate at which the system is monitored. If the frequency is too high the system may show excessive volatility while if it is not queried often the system may show no change if a change occurs. As a result, it is important to note that the frequency of querying will depend on a number of different factors including the metrics being monitored, the performance of the system, and the properties of the environment. It may be necessary for a monitorer to use

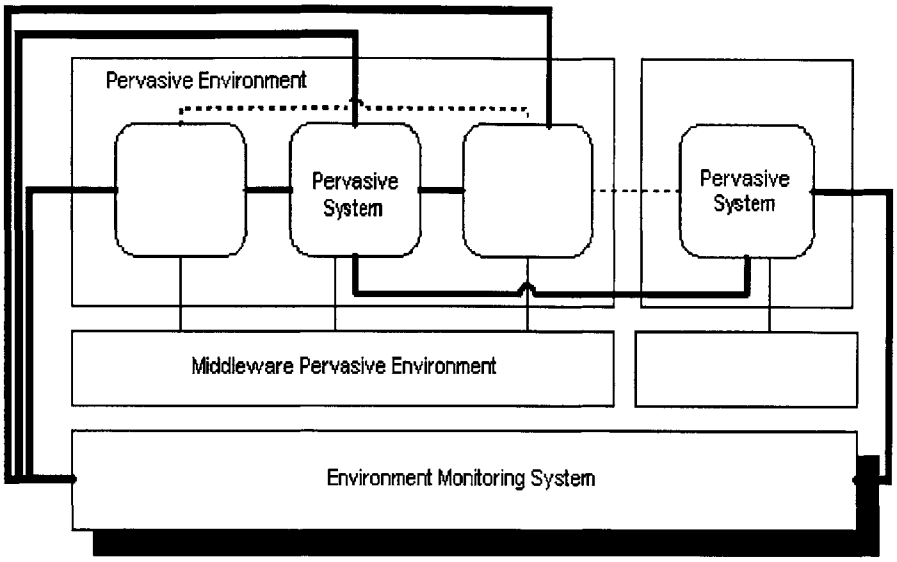


Figure 4-2: A system design for a pervasive environment. The system designers provide the process monitoring components for each process. The health monitoring monitoring device then queries these systems and processes the data

artificial intelligence techniques such as machine learning or clustering to correctly analyze the environment given the data it receives.

Additionally, since each process is being monitored, it gives the monitoring device the ability to monitor numerous things. If the monitoring device wants to explore only a few of the systems it can choose to do that, if the device wants to monitor the environment at the system level, it can group all the processes from a given system. This monitoring flexibility affords the monitoring device a great deal of freedom with which to explore the metrics within the environment. The described system design provides many of the characteristics that are necessary as described earlier in this thesis.

4.4 Other Metrics to Explore

Given these metrics there are many other areas to explore within this space. This section will describe a few of these methods including clustering and streaming data analysis.

In a similar vein to the distinct element counter, clustering techniques provide means to more information about a system. By clustering the data that passes through a communication channel one can get a better idea of the system's performance. If data starts arriving into a system that is not close to any of the current data clusters, it could indicate a change in the system's behavior.

There are numerous clustering algorithms that can be used to divide data up into meaningful clusters, based on size requirements of the clusters, or the distance between elements in each of the clusters. Thesis the various clustering algorithms that can be used will not be discussed but rather a focus on how to use clustering methods to aid in discovery of the properties of a system will be presented. For further references regarding clustering see Fasulo's analysis on recent work in clustering [12] or Berkhin's survey of data mining techniques [4]. A comprehensive look at clustering and machine learning techniques can also be found in Russell and Norvig's book [31].

Clustering data that passes through a communication channel gives an idea of the range of communication that occurs in a system. As a new cluster is created or eliminated within a system, it provides understanding on how the system is changing, if it is changing at all. Furthermore, while the transmission rate analysis and transmission data analysis metrics give values by which to quantify the communication channels, clustering allows a view of the data from a less rigid, numerical perspective to identify how elements are related on more than a single property.

There has been much work done on streaming data analysis [23], mostly stemming from the database community where there is often a need to analyze large databases in minimal time. The algorithms used in this streaming data analysis also have relevance to pervasive environments. In large database analysis and pervasive environments the analytical tool is only allowed one look at the data, and furthermore the tool is required to run with minimal storage and processing requirements. As such, many database algorithms are useful for pervasive environments and are possible areas to explore to gain further understanding of the systems. Additionally, if applied to the individual metrics, clustering and database analysis techniques may provide an interesting method by which to interpret these metrics.

Chapter 5

Conclusion

This thesis has outlined the background, needs, assumptions, and arguments in favor of identifying change points and more importantly keeping track of system metrics within a pervasive environment. As has been shown, a rich amount of data can be gathered with only a limited amount of storage or processor requirements on the individual systems within the environment. This gathered data has proven to be effective in understanding the status and performance of a system.

This thesis explored two interesting metrics for exploring the performance of systems in an environment: the rate of transmission of information and the actual information that was transmitted. Both of these metrics provided interesting and unique information about the environment and about the individual systems. This information can subsequently be used to accomplish many things including having systems understand how the other systems in their environment are performing, understand what state the systems are in, or simply gauge how the environment is performing.

Another advantage that these systems give is the ability to be used in conjunction with each other. The metrics have been built to be low cost in both storage and processing power, thus allowing them to feasibly be used in conjunction with each other. Additionally, the metrics were designed with the idea that the systems themselves would provide the metrics whenever queried. This is a powerful idea that permits the system designer to participate in the system in what ever way is most feasible

given the design of the system. This allows the system to be platform, hardware, and environment independent, as opposed to other monitoring systems. Additionally, under the assumption that the environment will be a system of systems, there is the idea that no central server exists. Many pervasive environments have a central server from which communication and coordination begin from. This environment does not operate under the assumption that a central server exists.

Furthermore, this thesis also went on to explore the mechanisms behind an out of band storage device that would monitor each of the systems and provide some notion of the performance of the environment while remaining independent of the actual system. Such an out of band system design would separate the monitoring and execution purposes within the environment and would create an environment within which to accurately and effectively monitor the systems. Furthermore, this system would allow users to easily consume the data in an understandable format and would provide a common monitoring point for the environment.

While this thesis has explored many of the metrics and opportunities surrounding pervasive environment analysis and subsequent user level debugging, there are still many areas that can be explored for further analysis. The subsequent section will elaborate further on other areas within which to explore on this current track.

5.1 Problems During the Course of Work

There were various problems during the course of this work. The primary problem was attempting to understand what window of time slices to use. Finding out this information was extremely problematic as different systems had different performance cycles and would show different results. Finding out these window sizes to lend relevant results was probably the most problematic situation we ran into.

Another area of work that did not lead to good results was attempting to use clustering methods as a performance metric. The implementations of clustering used on the distinct elements did not lend useful information and a different, thorough implementation of clustering may be necessary. The implementation that was used

once again attempted to minimize processor and storage capabilities, making it harder to optimize and create multidimensional clusters.

Finally, another major problem that was encountered was attempting to modify existing systems. As this implementation depends on having the system designer actually modify the system code to implement the API, without access to the system code to implement this API, many of the tests had to be run using external monitoring implementations that only served to increase the number of problems in the environment.

5.2 Future Work

The primary area that has been explored is in introducing new metrics such as clustering or machine learning techniques. These techniques were explored in very brief detail earlier in this thesis, but other analytical areas with promise can be found in database analysis or in other dynamic statistical analysis work. Some of these other techniques may include monitoring peak to peak times in the transmission rate analysis; observing if systems are communicating via RPC, stream, or other form of transmission; or additionally understanding the topology of an environment based on the transmission properties.

Another area that can be explored relates to the system design aspect of this work. A single system was designed to function with this metric structure. Other techniques that can be explored are methods by which system design can have a different topology. Furthermore, depending on the structure of the transmission there are circumstances by which metrics will not give a clear indication of the performance of a system. Often queries may happen in the middle of a transmission cycle or too far apart to have relevance. In such scenarios a different query system needs to be explored to fully understand the scope of the system. Designing this query system and the subsequent monitoring analysis tool is another area with plenty of future work.

A final area that could be discussed is the idea of monitoring transmission data

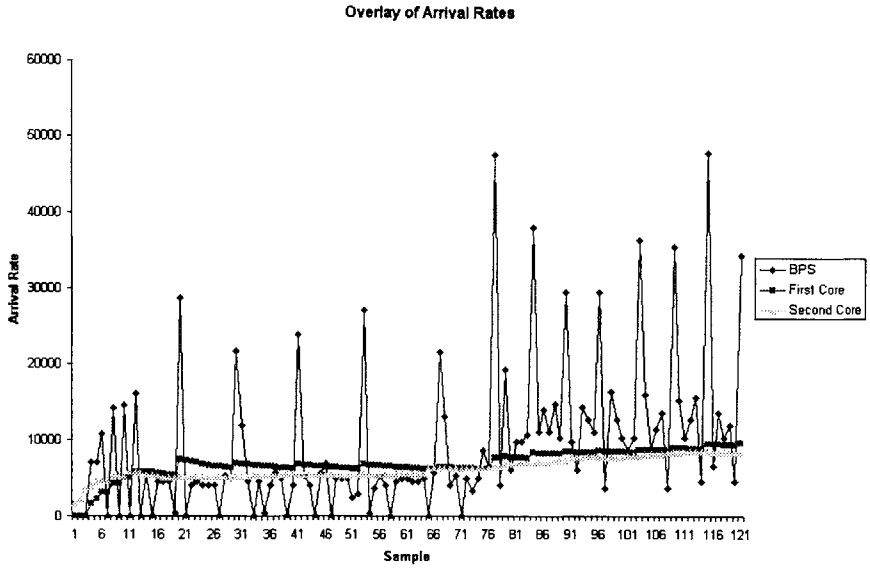


Figure 5-1: This graph shows an overlay of data leaving a CORE device and travelling to another CORE device. The data gives indication to the causality of the transmission.

to find correlations among different transmission streams. This subject was broached during the discussion of transmission rates in chapter 3.1, however, no work was done in this area for the scope of this thesis.

As can be seen in figure 5-1, where two COREs were setup between a VNC server and client setup, there is a clear causal relationship between the data sent by the first CORE and the data received by the second CORE. With no information about the structure of transmissions in an environment, it would be easy to monitor all the transmissions in an environment and correlate these transmissions with each other to determine where a specific transmission is going and what the latency of this transmission is. This would give more information about the structure of the environment and would allow a *monitorer* to learn a good deal about the structure of an environment.

Such techniques would be useful in exploring more monitoring techniques for pervasive environments, and building upon the work of this thesis.

Bibliography

[1] Mobisaic. http://www-cse.ucsd.edu/users/voelker/mobisaic/mobisaic.html.

[2] Cambridge AT&T Laboratories. Virtual network computing. http://www.uk.research.att.com/vnc/index.html.

[3] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream.

[4] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[5] Lawrence J. Brunsman. The application and design of the common oriented routing interface. Master's thesis, Massachusetts Institute of Technology, 2003.

[6] Ali Axarbayejani Christoper Wren, Trevor Darrell.

[7] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metaglue system. In Paddy Nixon, Gerard Lacey, and Simon Dobson, editors, *1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, pages 201–212, Dublin, Ireland, December 1999. Springer-Verlag.

[8] Graham Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *PODS 2003.*

[9] Mark E. Crovella and Thomas J. LeBlanc. Performance debugging using parallel performance predicates. In *Proceedings of ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 140–150, San Diego, California, 1993.

[10] G. Olson D. Agarwal and J. Olson. Collaboration tools for the global accelerator network.

[11] Glen Eguchi. Extending core for real world applications. Master's thesis, Massachusetts Institute of Technology, 2003.

[12] D. Fasulo. An analysis of recent work on clustering algorithms, 1999.

[13] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. In *Journal of Computer and System Sciences*. Winchester, United Kingdom, 1985.

[14] Krzysztof Gajos. Rascal - a resource manager for multi agent systems in smart spaces. In *Proceedings of CEEMAS 2001*, 2001.

[15] Armando Fox George Candea. A utility-centered approach to building dependable infrastructure services.

[16] Emily Yan Hana Kim, Nancy Kho and Larry Rudolph. commanimation: Creating and managing animations via speech.

[17] Nicholas Hanssens, Ajay Kulkarni, Rattapoom Tuchinda, and Tyler Horton. Building agent-based intelligent workspaces. In *ABA Conference Proceedings*, June 2002.

[18] Andrew C. Huang, Benjamin C. Ling, and Shankar Ponnekanti. Pervasive computing: What is it good for? In *MobiDE*, pages 84–91, 1999.

[19] Rebecca Isaacs. Performance analysis loosley-coupled distributed systems: the case for a data-driven approach.

[20] David Johnson and Willy Zwaenepoel. Recovery in distributed systems using optimistic message logging and checkpointing, 1998.

[21] Michael J. Katchabaw, Stephen L. Howard, Hanan L. Lutfiyya, Andrew D. Marshall, and Michael A. Bauer. Making distributed applications manageable through instrumentation. *The Journal of Systems and Software*, 45(2):81–97, 1999.

[22] Eric McDonald Lucy Dunne, Susan Ashdown. Smart systems: Wearable integration of intelligent technology.

[23] G. Manku and R. Motwani. Approximate frequency counts over data streams, 2002.

[24] Janet Wiener Patrick Reynolds Athicha Muthitacharoen Marcos Aguilera, Jeffery Mogul. Performance debugging for distributed systems of black boxes.

[25] Eugene Fratkin Armando Fox Eric Brewer Mike Chen, Emre Kiciman. Pinpoint: Problem determination in large, dynamic internet services.

[26] Erik Demaine Seth Teller Nissanka B. Priyantha, Hari Balakrishnan. Anchor-free distributed localization in sensor networks.

[27] Justin Mazzola Paluska, Jason Waterman, Chris Terman, Steve Ward, Umar Saif, and Hubert Pham. A case for goal-oriented programing semantics.

[28] David Patterson, Aaron Brown, Pete Broadwell, George Candea, Mike Chen, James Cutler, Patricia Enriquez, Armando, Emre Kcman, Matthew Merzbacher, David Oppenheimer, and Naveen Sastry. Recovery oriented computing (roc): Motivation, definition, techniques, and case studies.

[29] Manuel Román, Christopher K. Hess, Anand Ranganathan, Pradeep Madhavarapu, Bhaskar Borthakur, Prashant Viswanathan, Renato Cerqueira, Roy H. Campbell, and M. Dennis Mickunas. GaiaOS: An infrastructure for active spaces.

[30] Larry Rudolph. When rebooting is not an option. Technology Review.

[31] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2003.

[32] Joao Pedro Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments.

[33] Edward Tenner. *Why Things Bite Back : Technology and the Revenge of Unintended Consequences.* Vintage, 1997.

[34] Jeffrey S. Vetter and Patrick H. Worley. Asserting performance expectations.

[35] Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. The parctab ubiquitous computing experiment. Technical report, 1995.

[36] Mark Weiser. The computer of the 21st century, 1991.

[37] Peggy Wright. Knowledge discovery in databases: Tools and techniques, 1998.