SUIDS: A RESOURCE-EFFICIENT INTRUSION DETECTION SYSTEM FOR UBIQUITOUS COMPUTING ENVIRONMENTS

Bo Zhou

A thesis submitted in partial fulfilment of the
requirements of Liverpool John Moores University
for the degree of Doctor of Philosophy

June 2007

ABSTRACT

The background of the project is based on the notion of ubiquitous computing. Ubiquitous computing was introduced as a prospective view about future usage of computers. Smaller and cheaper computer chips will enable us to embed computing ability into any appliances. Along with the convenience brought by ubiquitous computing, its inherent features also exposed its weaknesses. It makes things too easy for a malicious user to spy on others.

An Intrusion Detection System (IDS) is a tool used to protect computer resources against malicious activities. Existing IDSs have several weaknesses that hinder their direct application to ubiquitous networks. These shortcomings are caused by their lack of considerations about the heterogeneity, flexibility and resource constraints of ubiquitous networks. Thus the evolution towards ubiquitous computing demands a new generation of resource-efficient IDSs to provide sufficient protections against malicious activities.

SUIDS is the first intrusion detection system proposed for ubiquitous computing environments. It keeps the special requirements of ubiquitous computing in mind throughout its design and implementation. SUIDS adopts a layered and distributed system architecture, a novel user-centric design and service-oriented detection method, a new resource-sensitive scheme, including protocols and strategies, and a novel hybrid metric based algorithm. These novel methods and techniques used in SUIDS set a new direction for future research and development. As the experiment results demonstrated, SUIDS is able to provide a robust and resource-efficient protection for ubiquitous computing networks. It ensures the feasibility of intrusion detection in ubiquitous computing environments.

ACKNOWLEDGEMENTS

My thanks go to the School of Computing and Mathematical Sciences, Liverpool John Moores University for their funding of this project. I would like to thank my supervisors, Dr. Qi Shi and Prof. Madjid Merabti for their help and 'fishing' me out from many applicants. In particular, I would like to thank Dr. Qi Shi for his trust, support, friendship, encouragement, inspiration and patience to correct my 'perfect' Chinglish. I would like to thank Prof. Madjid Merabti for his friendship and belief in me as a researcher.

My thanks go to the researchers and staff at Liverpool John Moores University for their support over the past years. I would like to thank (in no order): Steph Hogan, Carol Oliver, David Llewellyn-Jones, Mengjie Yu, Sareer Badshah, Tom Bailey, Gurleen Arora, Humayun Bakht, Bob Askwith, John Haggerty, Faycal Bouhafs, Henry Chang, Paul Fergus, Kashif Kifayat, Jennifer Tang, Ruth Thompson, Catherine Watts, Yuanyuan Shen, and Denis Reilly. In particular, my thanks go to David Llewellyn-Jones for his help and advice on Latex and Mengjie Yu for his friendship.

Finally, and by far most importantly, I would like to thank my family for their care, support, and trust over the years. Thanks to mum and dad. They created the opportunity for their son to study overseas. They kept belief in their son even when he was at the darkest moment. Thanks to my girlfriend, Daisy, for her patience and comfort despite my fussy temper under pressures. Without her understanding and care, this project would have never been completed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

With the continuous growth and development of computer networks, the notion of ubiquitous computing introduced by Mark Weiser has received increasing attention. However, this evolution faces a barrier. On the one hand, people want to construct a ubiquitous network to make the best use of computers; on the other hand, they must secure their network in order to cope with a number of security threats from malicious entities. One solution for this is to use an Intrusion Detection System (IDS). This chapter is organized as follows. First, the topic of the thesis is presented with its aims. Second, the novel contribution of the new approach posited in the thesis is presented. Third, an overview of the chapters of the thesis is presented. Finally, the chapter is summarized.

1.1 Background

With the wide spread of computers, our daily lives are highly computerised and closely connected with computer networks. In the near future, one will be able to open a door by simply sending an order to the electric door lock from his/her PDA, or read news on a computer embedded "e-paper" with the content updated through wireless connections [52]. The trend towards a computerised smart space is part of the conception of *ubiquitous computing* [1, 155]. In the era of ubiquitous computing, devices with computing and communicating abilities will surround us all over. Eventually it will achieve the non-intrusive availability of computers throughout physical environments. For example, hundreds of little appliances (e.g. computer embedded notes, pens and coffee machines) in a smart

office will be seamlessly integrated into a work environment, gently enhancing an occupant's everyday activities [47].

Just like other networks, one of the main prerequisites for a ubiquitous network is adequate security [32, 90, 142]. The network has to be properly secured so that it can be relied upon. *Intrusion Detection Systems* [4, 5, 31] are widely used to protect computer networks. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Moreover, an effective intrusion detection system can even serve as a deterrent, acting to prevent intrusions.

Traditional IDSs, which were originally developed for wired networks, are not suitable for ubiquitous computing due to the unique characteristics and inherent vulnerabilities of the environemnt. This unfitness directly compromises the effectiveness and efficiency of existing IDSs. For example, with the concept of ubiquitous computing, there must be some small-size devices in order to achieve unaware deployment. Inevitably, they will have limited energy supplies and storage spaces. An obvious issue is how to implement an IDS in a resource-effective way [94]. This is a big challenge since one of the most desirable features for an IDS is real-time detection and response, which is extremely energy consuming. Another key issue is related to the system architecture. Current host-based IDSs do not fit for ubiquitous computing due to the nodes' capacity constraints, while network-based IDSs simply cannot capture inside users' activities as the network's infrastructure tends to be heterogeneous. In chapter four these limitations are discussed in depth.

1.2 Project aims and objectives

The above discussion indicates that the evolution towards ubiquitous computing demands a new generation of resource-efficient IDSs to provide sufficient protections against malicious activities [105]. The aim of the project is to design such an IDS, which is able to minimize the use of system resources such as energy consumptions and communication overhead. It should have an appropriate system architecture and detection strategy to be flexible and scalable. The IDS must also be able to detect intrusions effectively, e.g. with a high hit rate

and a low false alarm rate.

The objectives of this project are:

- To provide a background to ubiquitous computing and demonstrate the unfitness of existing IDSs when applying them to ubiquitous computing environments.
- To posit the requirements for an appropriate IDS that is associated with resource-sensitive design and distributed modules' deployment.
- To present the design of a system (i.e. SUIDS, standing for Service-oriented and User-centric Intrusion Detection System) that detects attacks at the service layer and builds a defence wall against malicious users.
- To propose an original set of mechanisms, strategies and protocols that together achieve resource-efficiency in SUIDS.
- To prototype the SUIDS system in order to provide proof-of-concept for proposed work and perform an assessment in relation to the proposed requirements, where possible.

1.3 Novel contribution of this project

The key points of novelty of this project include:

- A layered and distributed system architecture, which is seamlessly embedded into ubiquitous computing environments. By categorizing system nodes into three major groups, SUIDS is more scalable and adaptable in order to fit for various network scenarios.
- A novel user-centric design and service-oriented detection method. By giving mobility to detection modules, SUIDS is able to react to malicious activities in real-time. It detects anomalies at the service level rather than relying only on the information from network layer.
- A new resource-sensitive scheme, including protocols and strategies. By allowing delegation of intrusion detection tasks to proxy nodes, SUIDS provides satisfactory intrusion detection service coverage to those nodes that are incapable of running IDS independently.
- A novel hybrid metric based algorithm. In order to balance system

resources such as CPU usage, network overhead, storage space, and energy consumption, SUIDS uses a hybrid metric to measure these factors together for the dynamic determination of cost-effective intrusion detection deployment. A node's trustworthiness is also considered in this hybrid metric to enhance the system's security policy.

- Critical assessment methods to evaluate the effectiveness and efficiency of the proposed IDS model. The effectiveness of SUIDS is reflected by its high hit rates on anomalies and low false alarm rates. Its efficiency is shown on deducted energy consumptions.

1.4 Project achievements

SUIDS is among the first intrusion detection systems proposed for ubiquitous computing environments. It keeps the special requirements of ubiquitous computing in mind throughout its design and implementation. The methods used in SUIDS set a new direction for future research and development. Practically, it ensures the feasibility and realization of intrusion detection in ubiquitous computing.

The outcomes of our research have generated the following conference and journal papers:

- B. Zhou, Q. Shi, and M. Merabti. Intrusion detection in ubiquitous computing environments. Proceedings of EPSRC Sixth Annual Network Symposium on the Convergence of Telecommunications, Network and Broadcasting (PGNet'05), Liverpool, UK, Jun. 2005, pp. 344-9.
- B. Zhou, Q. Shi, and M. Merabti. A novel service-oriented and user-centric intrusion detection system for ubiquitous networks. Proceedings of IASTED International Conference on Communication, Network and Information Security (CNIS'05), Phoenix, Arizona, USA, Nov. 2005, pp. 76-81.
- B. Zhou, Q. Shi, and M. Merabti. A framework for intrusion detection in heterogeneous environments. Proceedings of IEEE Consumer Communications and Networking Conference (CCNC'06), v2, Las Vegas, USA, Jan. 2006, pp. 1244-8.

- B. Zhou, Q. Shi, and M. Merabti. Real-time intrusion detection in ubiquitous networks with a string-based approach. Proceedings of IET International Conference on Computational Science and its Applications (ICCSA 2006), Part4, LNCS 3983, Glasgow, UK, May 2006, pp. 352-9.
- B. Zhou, Q. Shi, and M. Merabti. A survey of intrusion detection solutions towards ubiquitous computing. Proceedings of 1st conference on Advances in Computer Security and Forensics, Liverpool, UK, Jul. 2006, pp. 31-40.
- B. Zhou, Q. Shi, and M. Merabti. Intrusion detection in pervasive networks based on a chi-square statistic test. Proceedings of 30th IEEE Annual International Computer Software and Applications Conference (COMPSAC06), Chicago, USA, Sept. 2006, pp. 203-8.
- B. Zhou, Q. Shi, and M. Merabti. Balancing intrusion detection related energy in ubiquitous computing networks. Journal of Information Assurance and Security, vol. 1, issue 4, Dec. 2006, pp. 275-80.
- B. Zhou, Q. Shi, and M. Merabti. Resource-efficient intrusion detection in pervasive computing. Proceedings of IEEE International Conference on Communications (ICC 2007), Glasgow, UK, Jun. 2007, in press.
- B. Zhou, Q. Shi, and M. Merabti. Balancing intrusion detection resources in ubiquitous computing networks. Journal of Computer Communications, under review.

The simulation environment created in this project could be used as a primary testbed for other researches related to ubiquitous computing.

1.5 Thesis organisation

Chapter two: Chapter two reviews the history of computer networking and presents its future direction. Since the first set of computers connected together in the late 1960's, millions more computers have joined the network and formed an enormous cyber-world - Internet. Based on a layered architecture and well designed communication protocols, various computers on different platforms are able to communicate with one another under the same standard. In the near future, with the continuous growth and development of computer and network

technologies, it will enter the next stage of information era – ubiquitous computing. Smaller and cheaper computer chips will embed computing ability into any appliances, from a greeting card to a smart home. People's daily lives will be closely connected with computers and beneficially become ever convenient. Finally, along with the benefits, vulnerabilities of ubiquitous computing are discussed. Security is one of the major concerns for any computer network, including ubiquitous computing.

Chapter three: This chapter briefly introduces some attacks and countermeasures involved in computer security. In computer security, intrusions are defined as any malicious activities that could compromise the integrity, confidentiality, or availability of networks and information sources. There are many types of attacks on computer systems. As a second line of defence, IDSs play an important role in computer security, especially in the fight against attacks launched inside a network. The principles and classifications of IDSs are introduced in this chapter.

Chapter four: This chapter presents a critical survey on existing IDSs and the state of the art in intrusion detection related to ubiquitous computing. Although the research in intrusion detection started decades ago, its application to ubiquitous computing is new. Limitations and drawbacks of current IDSs are discussed. In particular, they cannot fulfil the special requirements of ubiquitous computing in respect of resource-efficiency and system architecture. An IDS in ubiquitous computing should not require transmitting or processing a large amount of audit data or attacking signatures. It should have a distributed or cooperative detection scheme instead of a centralized system architecture. In order to provide all-sided protection, resource constrained nodes in ubiquitous computing networks need special considerations for the design of an IDS. This chapter demonstrates the demand for a resource-efficient and robust IDS in such networks.

Chapter five: In this chapter, the system architecture and framework of our novel solution SUIDS are introduced. SUIDS is an adaptive and resource-efficient intrusion detection system with a novel service-oriented auditing mechanism and flexible user-centric design. In SUIDS network nodes are classified into three categories: head nodes, user nodes and services nodes. By working together with service-oriented (software) agents, SUIDS is able to reliably and effectively detect

malicious activities of inside users. It is suitable for heterogeneous environments such as ubiquitous computing networks. The simulation work of SUIDS is also provided in this chapter. As a research scenario, a smart home is simulated by using the Georgia Tech Network Simulator (GTNetS).

Chapter six: SUIDS is an anomaly-based intrusion detection system. Its detection algorithm and experiment results are presented in this chapter. To detect anomalies, SUIDS builds a profile for each user. The user's profile consists of the user's long-term behaviour, represented by his/her usage of service nodes. To achieve real-time detection, a string is utilized in the user profile to represent the user's short-term behaviour in due course. Every time a new event record arrives, the user profile is updated and the deviations between long-term and short-term behaviours are calculated. An appropriate string length and threshold value work together to balance the system's false alarm rate and detection effectiveness. With a carefully selected string length and threshold value, SUIDS can achieve a high hit rate while keeping the false alarm rate low.

Chapter seven: This chapter refines the detection method of SUIDS in order to improve its performance in terms of both detection effectiveness and efficiency. Instead of using a string, an exponentially weighted moving average (EWMA) technique is used to smooth out the observation value for the variables being tracked. In this way, the observation reflects the 'most recent past' characteristics of variables in an online fashion. It applies the smoothing constant in a user profile to represent the user's short-term behaviour in real-time. The deviations between a user's short-term behaviour and long-term profile are measured by using a chi-square statistic test. This method can measure not only the probability distributions of variables, but also their occurrence patterns.

Chapter eight: The inherent features of ubiquitous computing request SUIDS to give special concerns about the issue of resource-efficiency. This chapter presents a comprehensive analysis of energy consumed in SUIDS and proposes a profile splitting technique in order to reduce the energy consumptions. The energy consumed in SUIDS is categorized into two parts: computing-related and communication-related. Head nodes are used to save the computing-related energy. User profiles are managed in a distributed pattern to reduce the communication-related cost. A hybrid metric is used to measure multiple energy-related factors: transmission power, remaining energy, and energy

consumption rates. In order to balance energy consumptions among network nodes, proxy nodes can be selected based on the hybrid metric to share intrusion detection burdens with service nodes. In this way, the SUIDS achieves better performance in respect of energy-efficiency, so the network lifetime is beneficially extended.

Chapter nine: This chapter extends the work of chapter eight and takes more system resources into account during the selection of proxy nodes. Specifically, three deciding resources are considered in this chapter: energy, computing ability and trust level. A node's computing availability is measured by correlating with its energy usage. A faster energy consumption rate of the node means less computing ability available to intrusion detection. And its trust level is estimated based on multi-factors including its energy consumption pattern and 'safe time'. A new conditional hybrid metric is proposed in order to balance these limited resources together. The system's security policy is beneficially enhanced due to the consideration of nodes' trustworthiness.

Chapter ten: In this chapter, the requirements on IDSs in ubiquitous computing networks are reviewed. The performance of SUIDS is evaluated against these requirements. A successful IDS operating on ubiquitous computing networks must have the following five features: real-time detection, scalability and adaptability, full coverage, resource efficiency, and detection effectiveness. Comparing with existing solutions, SUIDS is the first IDS keeping the special requirements of ubiquitous computing in mind before its design and implementation. Specifically, SUIDS achieves real-time detection by giving mobility to its detection modules. The classification of network nodes and usage of lightweight agents make it scalable and adaptable. SUIDS considers capacity constrained nodes by adopting proxy nodes. Its novel hybrid metric balances multiple system resources, and in the meantime, it achieves high detection effectiveness while keeping the false alarm rate low. SUIDS provides a robust and resource-efficient protection for ubiquitous computing networks.

Chapter eleven: This chapter presents conclusions and future work.

1.6 Summary

This chapter has presented an overview of this thesis. In the near future, computing is becoming ubiquitous. Tiny embedded processes with the abilities of computing and communication will spread everywhere for the purpose of sensing, control and information display. Security protection, as an inevitable and critical issue, must be provided properly before the large-scale implementation and deployment of ubiquitous computing. Traditional IDSs are not fit for such an environment due to the resources constraints and heterogeneous infrastructure of ubiquitous computing. Therefore, this thesis provides a new and novel solution to the problem: SUIDS. This system adopts a flexible and adaptive system architecture to provide resource-efficient security protection against malicious activities. In the next chapter, the history of computer networks and the trend towards ubiquitous computing are introduced.

CHAPTER TWO

THE ROAD TOWARDS UBIQUITOUS COMPUTING

Many years after the invention, computers were supposed to work alone, running their own programs locally. This situation has changed in the late 1960's. A set of computers were connected together to allow remote access to computer resources. Since then, the world witnessed one of the greatest miracles in human history - Internet. In this chapter, we first introduce the history of computer networks, and then talk about the network structure underneath. Just like a language in human society, TCP/IP enables various computers and network hardware and software to communicate with each another under the same standard. Soon, with the continuous growth and development of computer and network technologies, we will enter the next stage of information era – ubiquitous computing.

2.1 A brief history of computer networks

In computer science, a *network* can be defined as a system for connecting computers by using transmission technology [23]. In the early time, computers were huge and expensive. They were hardly moved due to their large footprint. The computers' computing abilities were also limited. Programs took a long time to run. At that time, computers were 'luxuries' and only deployed in the top research labs.

In late 1960's, ARPA (Advanced Research Projects Agency) initiated a project aiming to connect researchers with computers [129]. The objective of the project was to enable researchers to remotely access to expensive computer

resources. As a result, ARPAnet - child of the ARPA project - became the first prototype of modern networks. Although it used a connection of only 50 kbits/s, ARPAnet brought a fundamental change from centralized to distributed computing and incorporated features of reliability and robustness, e.g. multiple links and distributed routing.

Computer networking is very complex. Before ARPA, there were many different hardware and software technologies from wired to wireless and from undersea optical fibber to home used infrared. No one technology is appropriate for every scenario. The occurrence of TCP/IP protocol glues together networks of many dissimilar technologies with routers [24, 146]. TCP/IP were developed in late 1970s and ARPAnet switched to TCP/IP in early 80's. The first switchover occurred in 1983 and it is regarded as the start of one of the greatest inventions – Internet.

Internet is defined as a set of networks connected by routers that are configured to pass traffic among any computers attached to networks in the set [23]. The global Internet is growing exponentially since its advent [22]. Initially the Internet had only a few hundred computers and a few dozen sites. Today, millions of computers and thousands of networks world-wide are connected together. No one knows the exact size of the Internet [34]. The recent ISC (Internet Systems Consortium) domain survey shows the growth of Internet in the past decade.

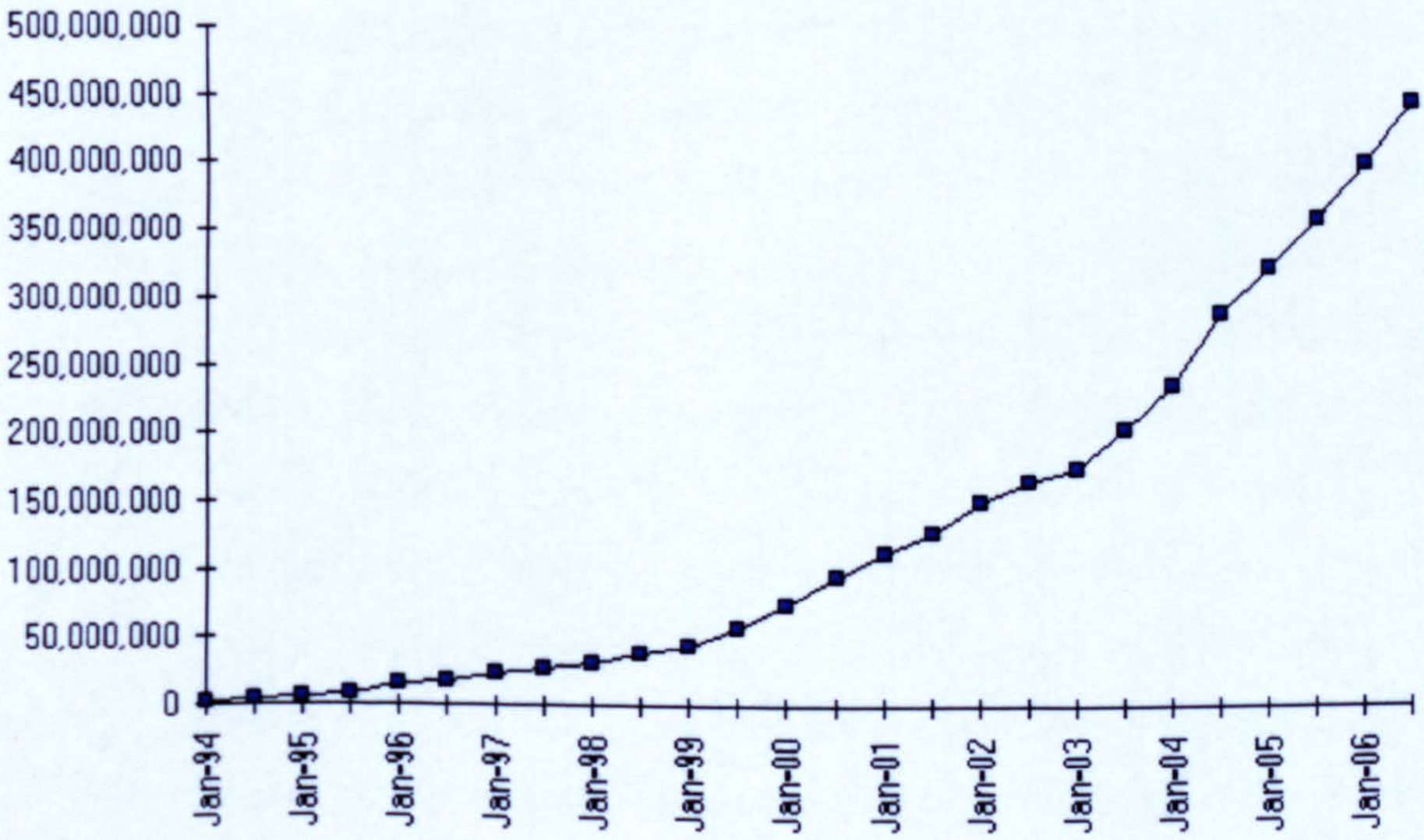

Fig. 2.1: Number of Internet hosts (Source: www.isc.org).

Internet also brought a new industry. Companies like Cisco [21], IBM [71] and Microsoft [104] continuously work out new products on networking hardware, computers and relevant software. Today, Internet has become a new phenomenon that networks are an important part of everyday activities. Through Internet, we can do shopping at home, finish a degree without going to an university, and make friends with people from anywhere of the world ... In many ways, it changes the way we live.

2.2 Network architectures

Hardware alone can't solve all computer communication problems. Software for Local Area Network (LAN) and Wide Area Network (WAN) systems provides high-level interface to applications. Standards have been adopted to allow a heterogeneous group of computers using a multitude of operating system software to communicate with one another. The layering model is a well known structuring solution to organize the complex networking software design and implementation. A layer on one machine communicates directly with the corresponding or *peer layer* on another machine to which it is connected. The rules and conventions which allow this communication to take place are enforced through *layer protocols*. They specify the format and meaning of messages exchanged between computers across a network. The set of protocols used in the communication between systems provides a *network architecture*.

The International Standards Organization's (ISO) Open Systems Interconnection (OSI) 7-layer reference model is the most widely-used model [30]. It is a guide to the design of a network protocol suite. Layers are named and numbered from bottom to top as shown in Fig. 2.2. Each layer fulfils specific functions in the communications between the two computers. The application layer consists of a number of protocols that are commonly needed, for example, File Transfer Protocol (FTP) [117] and Simple Mail Transfer Protocol (SMTP) [116]. The presentation layer defines the common formats for the representation of data. The session layer manages sessions such as login to a remote computer. The transport layer is designed to let computers carry on a conversation and is the

heart of the whole protocol hierarchy [149]. Two types of transport service, i.e. the connection-oriented Transmission Control Protocol (TCP) and the connectionless User Datagram Protocol (UDP), work at this layer to ensure the reliable delivery of data between computers [24, 25, 146, 149]. The network layer is in charge of address assignment and data delivery across a physical network by using the Internet Protocol (IP). The data link layer formats data in frames and delivers frames through a network interface. The physical layer includes basic network hardware such as RS-232 or Ethernet [24, 25, 146, 149].

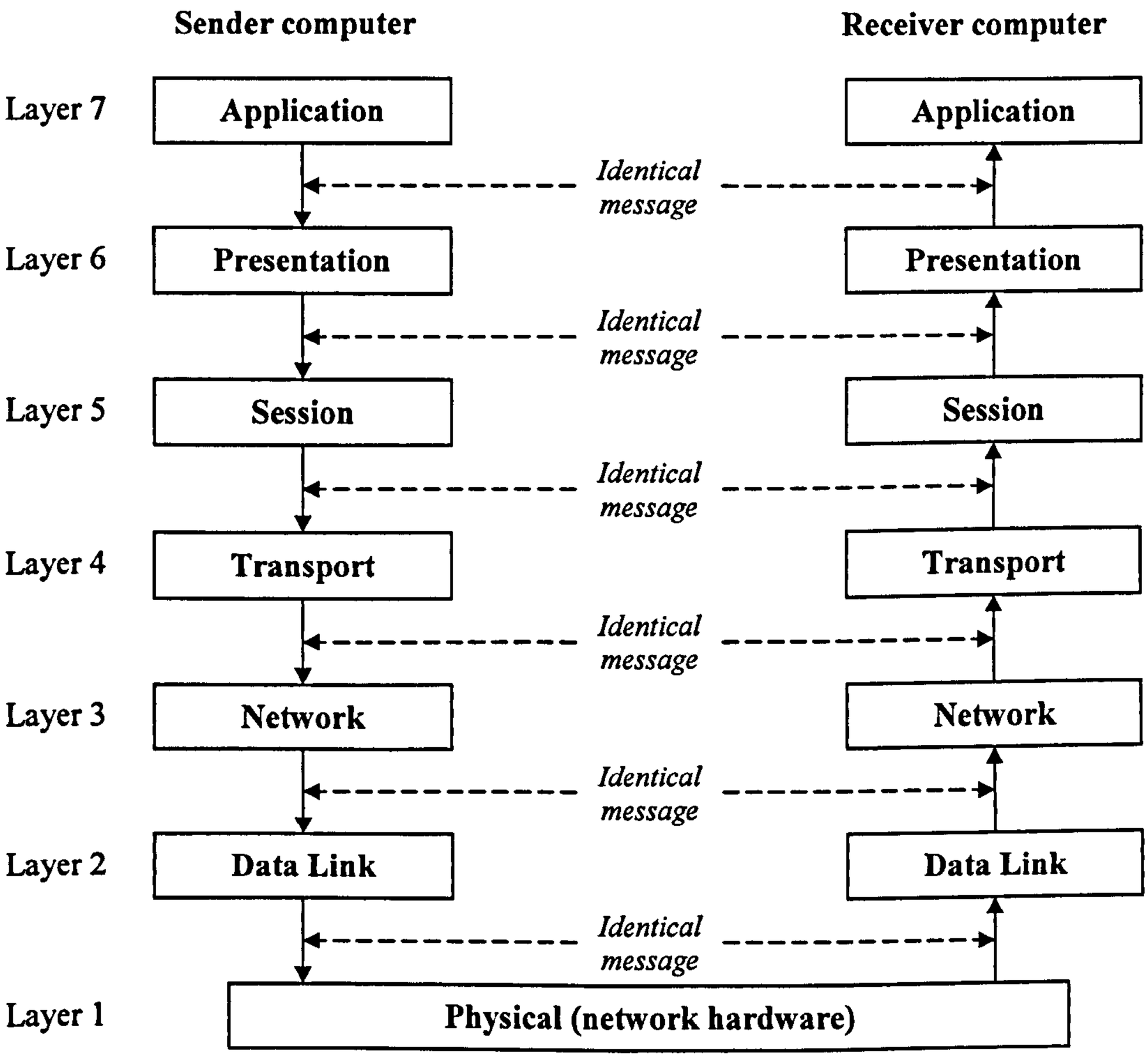

Fig. 2.2 ISO OSI 7-layer reference model.

IP is an unreliable, connectionless delivery service, so there are no guarantees that an IP datagram will reach its destination [24, 91]. In IP, data are transmitted in small and independent pieces, i.e. packets or datagrams. Each packet placed on

the network will be automatically routed through a number of networks until it reaches its destination. Packets travel independently and may follow different paths based on the network's status. The source divides outgoing messages into packets and the destination reassembles received packets to get the original data. Packets may be delivered out of order, especially in systems that include multiple networks. It can be detected and corrected through sequencing. The sender attaches a sequence number to each outgoing packet and the receiver uses the sequence number to put packets in order and detects missing packets. Lost packets perhaps are the most widespread problem. Any error such as a bit error or network congestion may cause a packet to be discarded or undelivered. Protocols use positive acknowledgments with retransmissions to detect and correct lost packets. The packet receiver sends a short message acknowledging receipt of packets. The sender sets a timer for each outgoing packet and infers lost packets from missing acknowledgments. If a timer expires before the acknowledgment is received, the sender will retransmit the lost packets.

The flexible layered architecture allows multiple networks and computers to connect in a seamless way, irrespective of the requirements demanded by various applications. Software implemented from the layered design has layered organization. The software for each layer depends only on the services of the software provided by lower layers. The software at layer *n* at the destination receives exactly the same protocol message sent by layer *n* at the sender (Fig. 2.2). It means the protocols can be tested independently and replaced within a protocol stack. The software at each layer communicates with the corresponding layer through information stored in headers. Each layer adds its header to the front of a message from the next higher layer. Headers are nested at the front of the message as the message traverses the network (Fig. 2.3). On the sender side, each layer accepts an outgoing message from the layer above, adds a header and other processing information, and passes the resulting message to the next lower layer. On the receiver side, each layer receives an incoming message from the layer below, removes the header for that layer, performs any other processing, and passes the resulting message to the next higher layer.

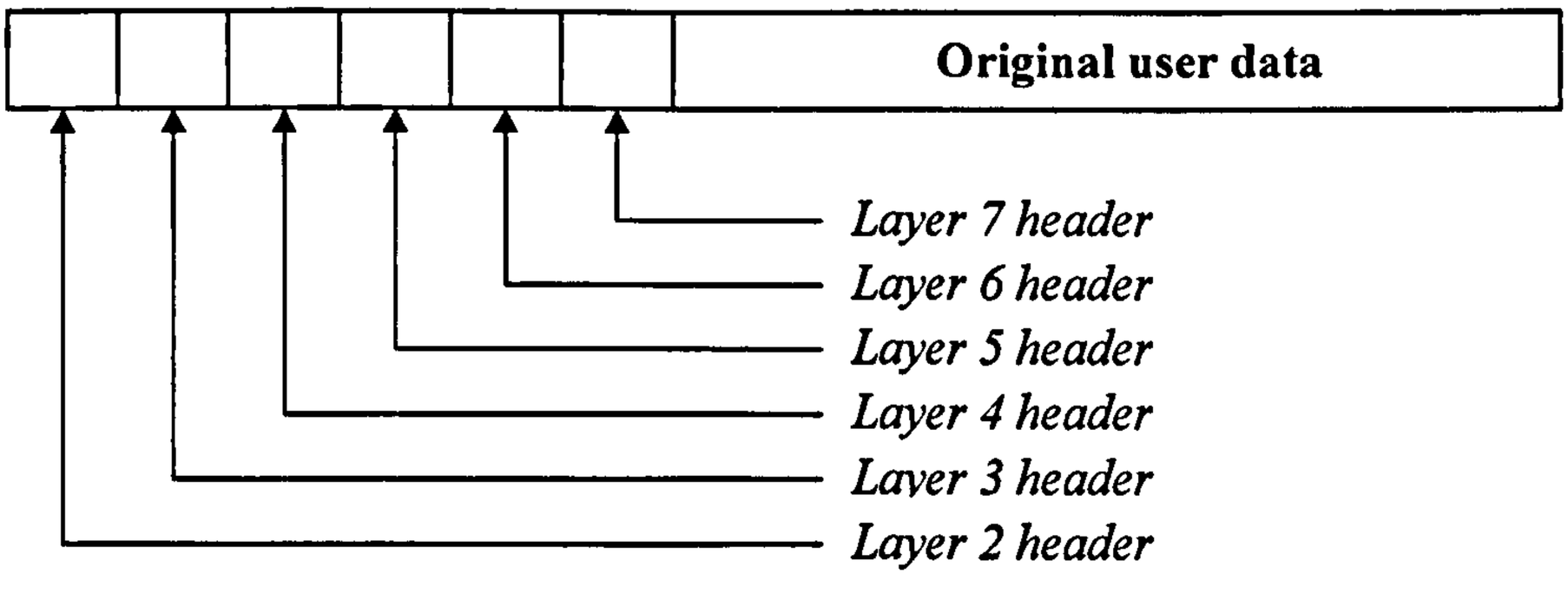

Fig. 2.3 Nested layer headers.

2.3 Concept of ubiquitous computing

The term of *ubiquitous computing* was brought out by Xerox PARC (Palo Alto Research Center) in 1991. It was first mentioned in Mark Weiser's article "The Computer for the 21st century" [155]. The author explained that the most powerful and successful technologies are those that naturally blend into our world until they are effectively invisible. These technologies become human's second nature due to their usefulness and wide availability. People stop thinking of themselves as using a technology; instead, they just consider themselves capable of doing whatever the technology enables. A good example is the telephone [32]. If people say "I spoke to my brother in London this morning", we understand implicitly that they used the telephone networks to do so. We would never hear someone saying like "This morning I used the telephone networks to speak ..."

Just like the telephones, computing is becoming ubiquitous as well. Five trends indicate the technical feasibility of this change [102, 147]. The first trend is given by Moore's law [106]. It states that the number of transistors on the same chip area doubles every eighteen months. Thus, the price of computer chips are getting lower and their sizes are getting smaller. It makes integrating computer chips into daily items become possible. The second trend is the emergence of new materials. Smart paper [40][45] for example provides a new interaction scheme with IT systems: a thin and flexible plastic foil contains the electronic ink that can display information as well as be used as an input device with a special pencil. Progress in communication technologies dominates the third trend. Except for

higher bandwidth, mobile networks like mobile phones or mobile ad-hoc networks have been widely deployed in recent years. No matter sitting in a café or airport lounge with a small PDA, people can easily be kept updated of their business. Progress in sensor technology is the fourth trend: like computer chips, sensors are getting smaller and cheaper, so that they can be integrated into everyday items to observe surrounding environments. A milk bottle for example could calculate its expiry date depending on its current temperature: the colder the milk bottle, the later the expiry date. The last trend refers to new concepts that model the infrastructures for these smart everyday items. People reinforce existing devices with computers because they are more effective, well-understood, and reliable. Actually we always choose the most comfortable technologies even when alternatives exist. That is why lights and doorbells are all operated by electricity now.

An online medicine cabinet is a good illustration to understand the notion of ubiquitous computing [47]. Imagine that you are walking into the bathroom in the morning. Your medicine cabinet recognises you and tells you that you should take your allergy medicine since it is a high pollen day. Because the cabinet knows your needs, it will gently warn you if you pick up a wrong drug. If you are almost out of pills, the cabinet will automatically order them online and refill it.

Several components form such an online medicine cabinet:

- A basic computer system. The cabinet must be able to store information such as the user's health condition and the functionalities of medicines.
- A context-aware mechanism. The cabinet must be able to recognise the user and sense the type and availability of the medicines.
- A communication network. The cabinet should be able to receive the information related to the medicine (in this case it is the weather) and order the medicine automatically online.

The components listed above already exist, but they are typically conceived and operated independently in the context of their own restricted view of the world. Current research is focused on the problem of combining them together and creating integrated ubiquitous computing systems. Many devices will be networked together to provide portable, effortless access to a global information infrastructure. The concept of computing will no longer imply a workstation with a single display screen demanding its user's attention; rather, there will be a

collection of displays everywhere allowing casual, low-intensity use. Computing power, including data storage and retrieval, will be everywhere in the environment and the devices needed to access that power will be freely available like ball point pens, which you pick up to use as needed and then leave behind when you're done with them.

Additionally, ubiquitous computing will also have a great impact on today's business processes. When companies plan to adopt a new technology, they want to know business impacts in advance. These impacts are mainly characterized by costs and benefits. We already mentioned that the costs for computer chips are getting lower and lower. The benefits can be concluded as: the avoidance of media breaks between the real world and the digital world, the awareness of "smart objects", and the support for mobility.

The avoidance of media breaks means the potential to improve the efficiency and quality of business processes through automation. A high level of process automation leads to reduced cost since less human intervention is required and more human errors are eliminated. For example, all the goods in a supermarket can have an embedded small chip on their tags. It can record necessary information such as the price of the product and be sensed by exit doors. Customers can check out immediately without queuing at the casher and the stock count could never be easier and more accurate. The awareness means that objects are able to provide data about their current and past context. Decisions that affect an object can be made at the object itself. For example, a milk bottle can decide and be asked whether it was stored always at the right temperature. In a traditional process it must be ensured that thermometers are always around for external monitoring. These thermometers must be checked every time the milk bottle changes its location. This process is laborious and error-prone, and does not provide an appropriate means to measure the actual quality of the milk bottle.

However, along with the benefits, ubiquitous computing also brings numerous vulnerabilities. It makes things too easy for malicious people to build a system to spy on others. A basic concern about any information stored in a computer is who can access and modify the contents. Where are the bits? Are they secure? And more questions will be asked especially if the information is collected from environments and transmitted over networks. Although issues surrounding the appropriate use and dissemination of information are as old as the

history of human communication, specific concerns stem from the fact that ubiquitous computing makes information more generally available. Imagine, when a visitor uses your bathroom, you will not expect your medicine cabinet to leak your health condition out to him; when the cabinet buys the pills online, you will need it to keep your personal/financial information secure. The situation could become even more worrying if your medicine cabinet already has been compromised. What will happen if the cabinet advises you with wrong doses? And what will happen if the cabinet changes the medicines without your awareness?

The above discussion clearly suggests that a strong security mechanism is necessary to ubiquitous computing. In this thesis, we particularly pay attention to one of the most important security solutions - intrusion detection.

2.4 Summary

In this chapter, we introduced the history of computer networks. Since the first network project carried out in later 1960's, computer networks never stop growing. Till now, millions of computers have joined together forming the biggest cyber-society: Internet. The notion of ubiquitous computing was introduced as a prospective view about the future usage of computers. Smaller and cheaper computer chips will enable us to embed computing ability into any appliances, from a piece of paper to a racing car. People's daily activities will be closely connected with computers and beneficially become ever convenient. However, the great features of ubiquitous computing inevitably expose its inherent vulnerabilities. A ubiquitous network must be properly secured so that it can be relied upon. In the next chapter, we will present the work related to network security. As a defensive countermeasure, IDSs will be introduced.

CHAPTER THREE

NETWORK SECURITY AND INTRUSION DETECTION SYSTEMS

In computer security intrusions are defined as any malicious activities that could compromise the integrity, confidentiality, or availability of networks and information sources. For example, an attacker may compromise the availability of an information system by flooding a server with an overwhelming number of service requests over a short period of time or deliberately wasting the server's CPU time simply with a paragraph of malicious code. Another attacker may compromise the integrity and confidentiality of an information system by gaining the privileges of an authorized user and then modifying or stealing information. In this chapter, we briefly introduce possible attack types and countermeasures involved in computer security. As an effective tool against inside threats, the principles and classifications of intrusion detection systems are specifically discussed.

3.1 Computer security

Computer security is a subfield of computer science, regarded as the control of risks related to computer usage. A traditional approach to coping with this issue is to specify and enforce a security polity on a computer system to restrict the actions an entity (user or program) can perform. There is no universal standard defining what secure action is. A university may have a very different notion of security from a military base. Thus security here is a property that is unique to each situation and so must be overtly defined by a security policy.

A secure system should still permit authorized users to carry out legitimate and useful tasks. One might be able to secure a computer beyond misuse using extreme measures such as those noted by author Eugene H. Spafford: *The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts* [35]. However, this would not be regarded as a useful secure system. There is always a trade-off between the security and utility of computer systems.

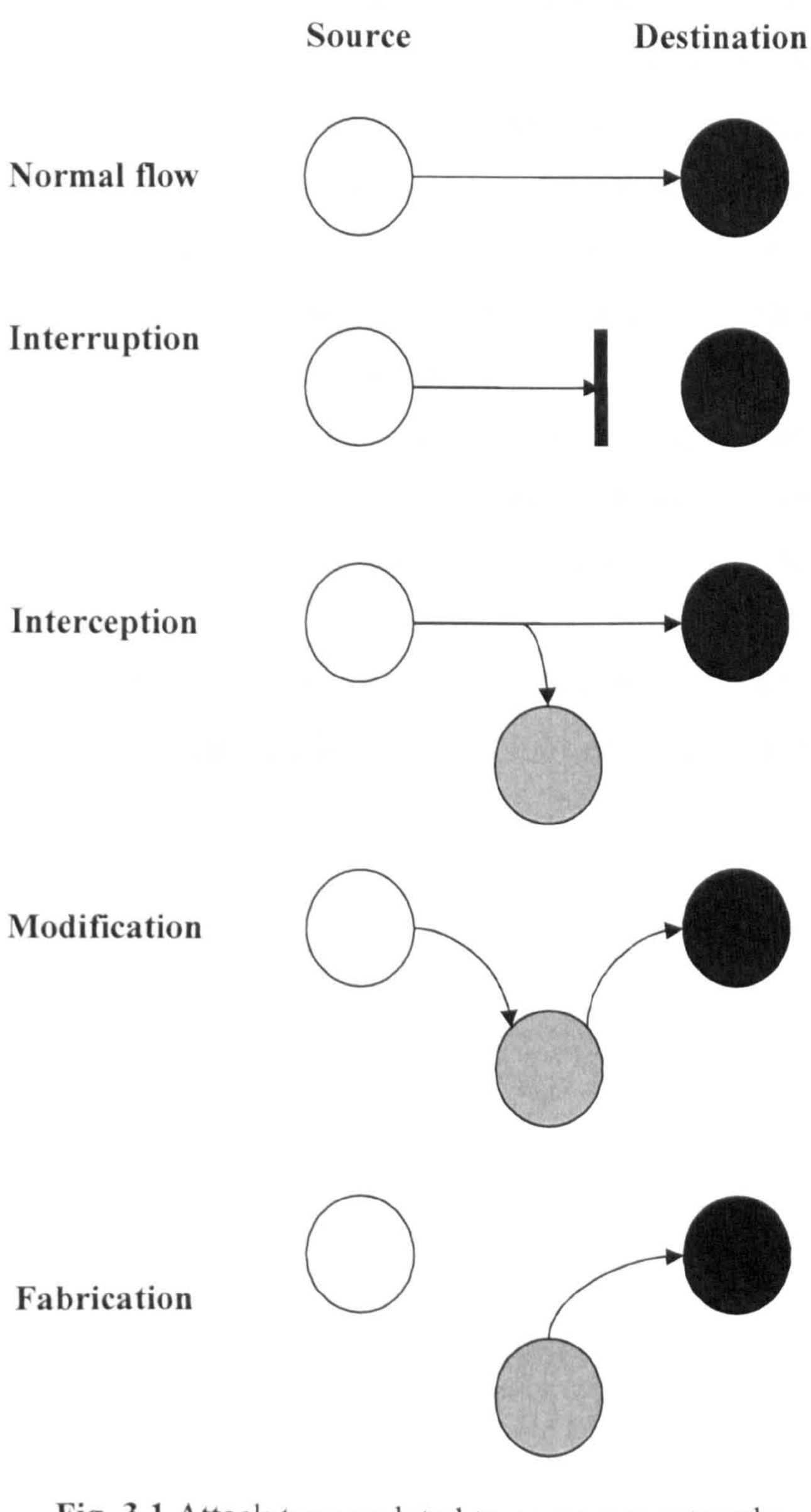

Fig. 3.1 Attack types related to computer networks.

With the advent of the Internet, tens of thousands of networks are connected together through routers. These routers forward packets from their sources to the destinations. It gives legitimate users, as well as malicious users, easier accessibility to the computer systems. And since then, computer security is no longer only about computer consoles, but also their connections with the outside world. Fig. 3.1 illustrates the possible attack types regarding computer networks [143].

A recent FBI survey (2005) suggests that the vast majority of organizations (87%) experienced some type of computer security incident [26]. More than 79% said they'd been affected by spyware and almost 84% were affected by a virus attack at least once a year, despite the almost universal use of antivirus software. The target of an attack could be any part of a computing system. Fig. 3.2 shows some examples of targets and attacks related to a computer system. Basically, a working security policy should include [59]:

- Data accessibility – the contents are accessible to legitimate users.
- Data integrity – the contents are not modified by unauthorized entities.
- Data confidentiality – the contents are not revealed to unauthorized entities.
- Accountability - responsible for tracking who has accessed the data.
- Authorization - responsible for who is allowed to access the data.

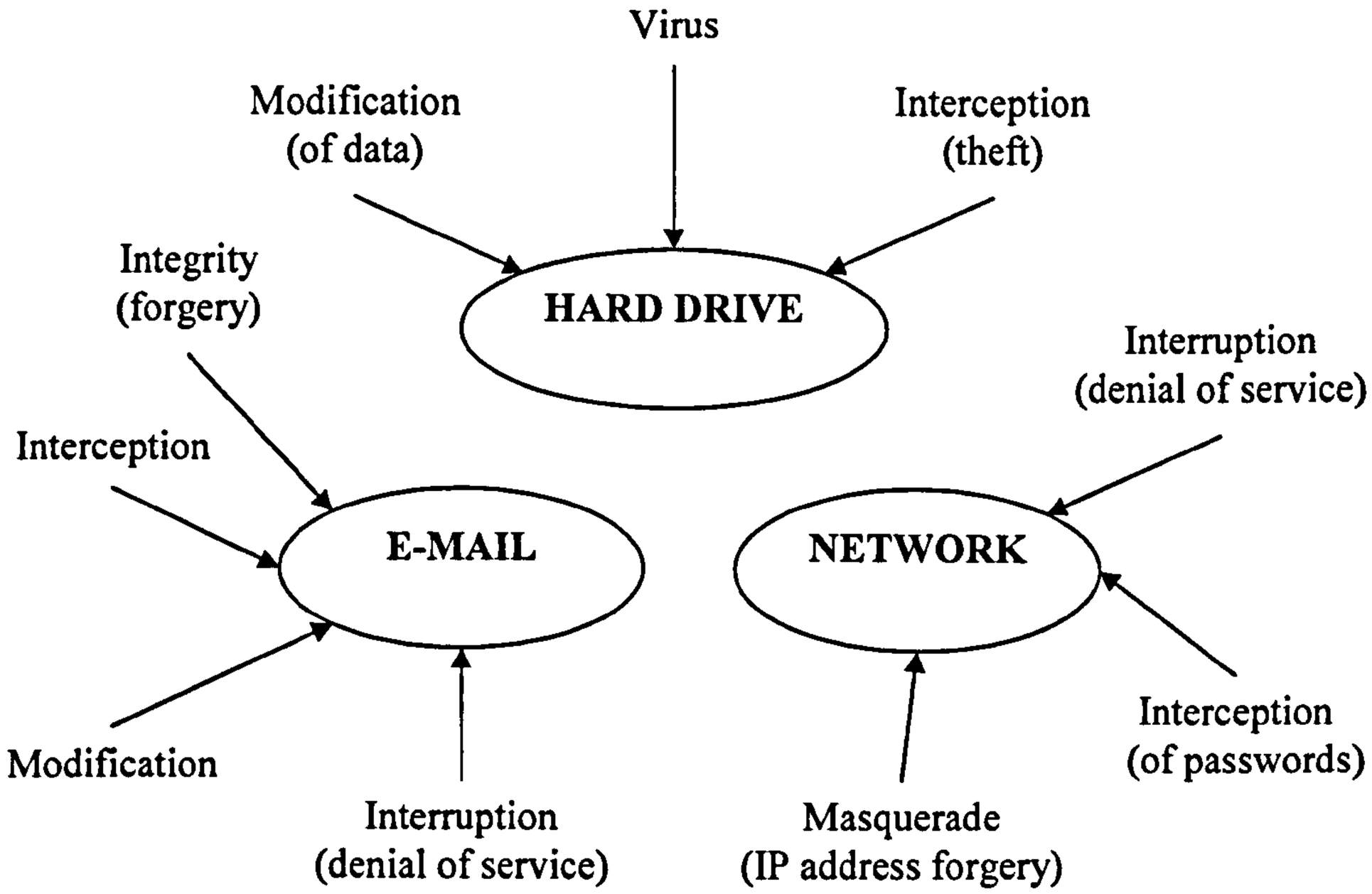

Fig. 3.2 Examples of attack types and system targets.

Countermeasures against security breaches work together at different network layers as shown in Fig. 3.3 [143]. Some of them are briefly explained here:

- CRC (Cyclic Redundancy Check): A type of hash function used to produce a checksum – a small, fixed number of bits – against a block of data, such as a packet of network traffic. The checksum is used to detect errors after transmission. A CRC is computed and appended before transmission, and verified afterwards by the recipient to confirm that no changes occurred on transit.
- Encryption: A mathematical procedure of rewriting contents so that they cannot be read without the corresponding decryption key. The encrypting function produces an encrypted message, while the decrypting function extracts the original message from the encrypted one. An encryption key is a parameter that controls encryption/decryption. A message sender and a receiver share a secret key for symmetric encryption/decryption. Key management is the crucial part of the encryption.
- Digital signatures: A public-key/asymmetric cryptographic method used for message authentication and integrity checking to deter fraudulent

activities. A sender encrypts a message with its private key and a receiver decrypts the encrypted message with the public key linked to the sender's private key. As this pair of keys should be uniquely associated with their owner and certified by a certification authority, the encrypted message can only be generated by the key owner. This guarantees that the message must be originated from the key owner.

- Firewall: A device located at the edge of networks to permit or deny data connections. Firewalls can be hardware and/or software based. A firewall's basic task is to control traffic between computer networks with different zones of trust and it is configured based on the organization's security policy.
- IPSec (IP Security): A framework operates at the network layer by extending the IP packet header (using additional protocol fields, not options). This gives it the ability to encrypt packets from any higher layer protocol, including arbitrary TCP and UDP sessions, so the information cannot be captured and understood by outsiders. It is widely used between two private networks over the Internet to support virtual private networking (VPN).
- Kerberos: A secure method for authenticating a request for access to a service in a computer network. Kerberos issues a user an encrypted 'ticket' in an authentication process, so the user can use the 'ticket' to request a particular service from an application server. The user's password does not have to pass through the network.
- PGP (Pretty Good Privacy): A free and widely used encryption program that lets user protect files and electronic mails. PGP uses both conventional and public key cryptography so it can be used to authenticate messages, protect their integrity, and keep them secret.

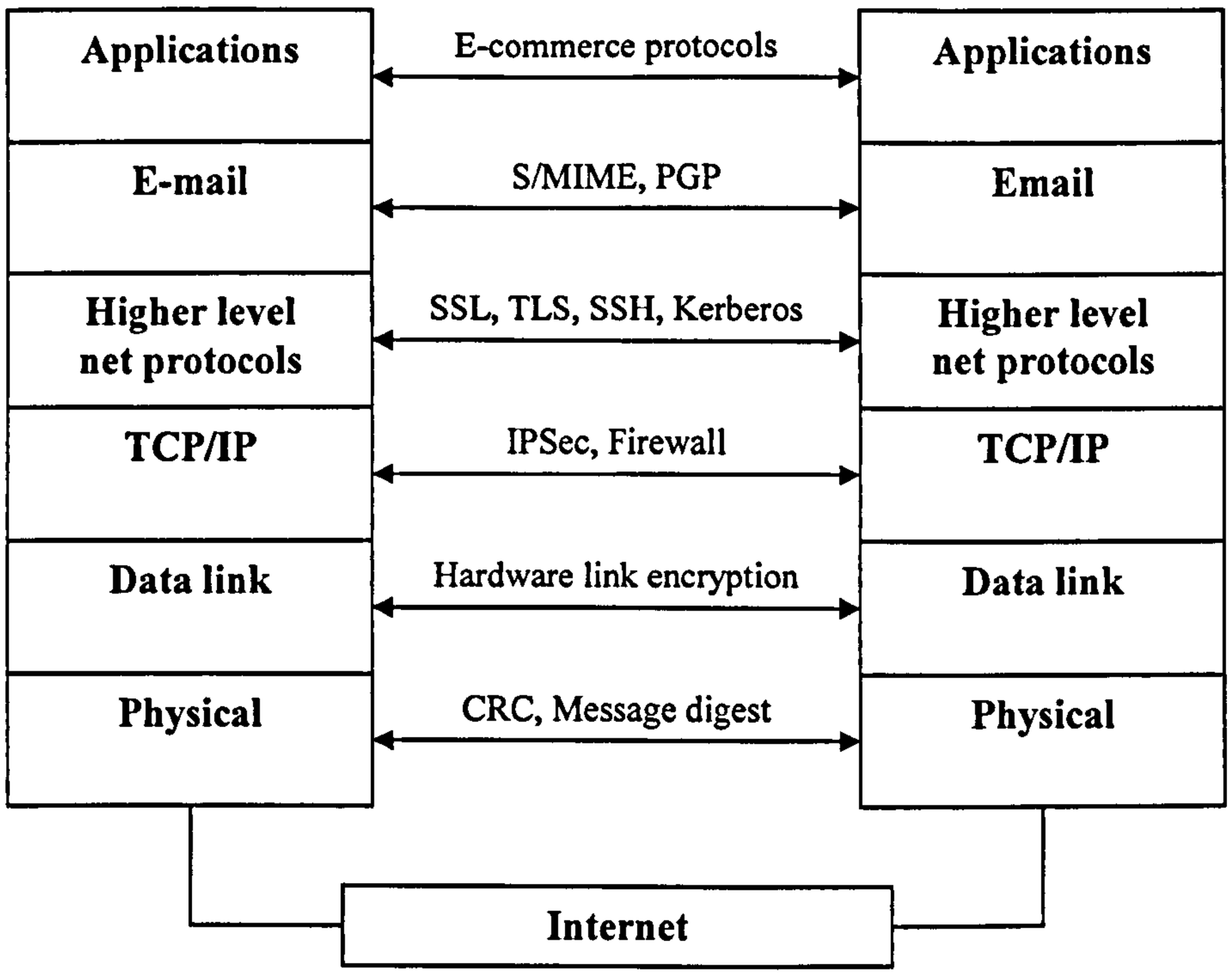

Fig. 3.3 Examples of security countermeasures at different network layers.

All the methods mentioned above are used to protect against attacks from the outside of an organization. However, the same FBI survey (2005) reported that 44% organizations had experienced intrusions from within their organizations. Further, the average cost of a successful attack by a malicious insider is much greater than the cost of an external attack. It emphasizes the needs for another type of security tool – Intrusion Detection System.

3.2 Intrusion detection systems

An IDS detects and makes alarms when intrusions have taken place or are taking place in a network being monitored. It achieves detection by continuously monitoring unusual activities happening in the network [4, 31]. The basic hypothesis of IDS is that there must be some trails connected with intrusions, at least traceable for a certain period. Unlike firewalls which are designed to prevent the occurrence of intrusions, an IDS only works after intrusions have occurred or

even succeeded. That is why an IDS is thought as the second line of defence. The main advantage of IDSs over firewalls is that IDSs can detect not only the attacks launched outside a network, but also inside attacks.

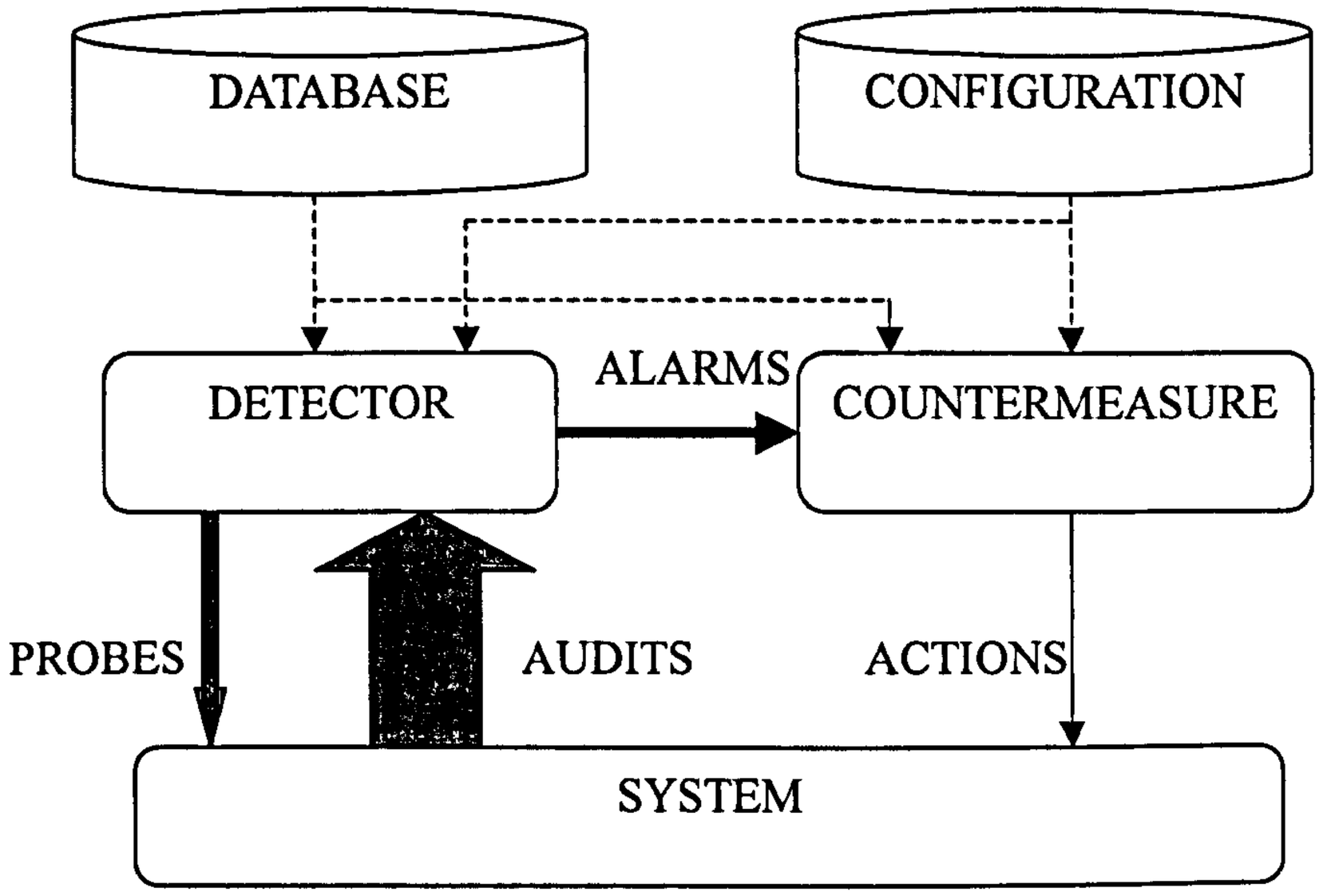

Fig. 3.4 A basic model of IDS. *Note: The arrow thickness represents the amount of information flowing between components.*

A basic model of an IDS is shown in Fig. 3.4 [4]. It includes quite a few components. Basically, intrusion detection decisions are made based on collected audit data. Audit data come from diverse sources, which could cover all the network layers and operating system states. The volume of traffic and required storage space for audit data can be huge, especially for long-term auditing. Detectors monitor the audit trails and execute one or more detection algorithms to find the evidence of suspicious actions. The database is used to store signatures (for signature-based detection, termed as known attacks or system vulnerabilities) or profiles (for anomaly-based detection, termed as reference models of usual behaviour). If any intrusion has been detected, the IDS will take certain response, for example, alert the system administrator or disconnect the suspected session. The IDS is controlled by the configuration of system settings that would specify how and where to collect audit data, how to respond to intrusions, and so on. Configuration is crucial because attackers could take advantage of improper configuration to bypass the intrusion detection. The system administrator is in

charge of setting an effective configuration.

There are two basic requirements for all IDSs: effectiveness and efficiency. Effectiveness means that an IDS must be able to correctly identify malicious activities from normal usage. Both false positive (indicating normal activities as malicious) and false negative (skipping malicious activities) are unwanted and must be kept under certain level. Efficiency means that an IDS must run in a cost-efficient way. Overhead introduced by an IDS on CPU usage, storage space and network resources confines its usability. The implementation of an IDS should not disturb others systems carrying out their normal activities.

3.2.1 Signature-based and anomaly-based IDS

According to the detection methods used, IDSs can be divided into signature-based detection and anomaly-based detection. Signature-based (also called knowledge-based or misuse-based) detection compares audit data with the knowledge accumulated about specific attacks and system vulnerabilities. General techniques include state modelling, expert systems, string matching and simple rule-based checking [31]. For example, a signature rule for a "guessing password attack" can be "there are more than 4 failed login attempts within 2 minutes". The main advantage of signature-based detection is that it can accurately and efficiently detect instances of known attacks. The main disadvantage is that it cannot detect unknown intrusions and a regular update is needed.

Anomaly-based detection builds a reference model of the usual behaviour of the system being monitored and looks for deviations from the normal usage [57, 88, 99]. Statistical methods have been used to detect anomalous network activities. For example, the normal profile of a user may contain the averaged frequencies of some system commands used in his or her login sessions. If for a session being monitored, the frequencies are significantly lower or higher than the normal usage, an anomaly alarm will be raised. Instead of simply measuring the means or variances of variables, SRI's NIDES [99] developed a more sophisticated statistical algorithm by using an X^2-like test to measure the similarity between short-term and long-term profiles. Neural networks are also widely considered as an effective approach to classify anomaly patterns. The paper [10] uses BP neural networks to detect anomalous usage of programs. The main advantage of anomaly

detection is that it does not require prior knowledge of intrusions and can thus detect new intrusions. The main disadvantage is that it may have a relatively higher false alarm rate.

Additionally, a number of IDSs adopt a hybrid system design, i.e. combine both signature-based and anomaly-based detection modules together. For example, in NIDES [99] a statistical model and an expert system were both used to detect intrusions.

3.2.2 Host-based and network-based IDS

According to the locations of audit sources, IDSs can also be categorized as host-based IDSs (HIDSs) and network-based IDSs (NIDSs) [4, 31]. HIDSs audit data are mainly from local operating systems, e.g. system log files. On the one hand, host audit sources are the only way to gather information about the activities of users on a given machine; on the other hand, they are also vulnerable to alterations in the case of a successful attack. This creates an important real-time constraint on HIDSs, which have to process the audit trail and generate alarms before an attacker taking over the machine can subvert either the audit trail or the intrusion detection system itself. HIDSs put higher requirements on individual nodes. The nodes in HIDSs have to dedicate a certain amount of resources to intrusion auditing, e.g. maintaining a large number of historical log files. Besides, the reliability of HIDSs is, to a great degree, determined by the accuracy of audit sources, but some devices may not be able to provide sufficient audit trails due to their oversimplified operating systems.

NIDSs overcome those issues by auditing network packets instead of system log files. NIDSs audit network packets between nodes or the Simple Network Management Protocol (SNMP) information. They do not require extra efforts from normal network nodes except for those running detection modules. However, the efficiency of NIDS is under suspicion [119] and the allocation of detection modules also became a controversial issue [13, 105, 161]. Most existing NIDSs are implemented on network devices such as routers and switches. They adopt a sniffer-based technique to gather the network traffic they need. Sniffers placed in front of a switch or router will see all the IP packets on a subnet. However, considering the increasing diversity of network infrastructures, a user's activities

within the network may not be noticed by the network devices. For example, when a user opens an electric door, he might use his PDA to send a login request to the door lock. It is very likely that the request will not be captured by any network devices due to its limited propagation range. This could give the inside user opportunities to bypass the network intrusion detection. Recent researches have already shown that the primary threat comes from individuals inside organizations as inside attacks are more damaging than attacks launched outside [16, 26, 150].

Hybrid approaches have also been developed using both network-based and host-based intrusion detection tools in a multi-host environment, i.e. a network of workstations. For example, DIDS [138] detects local attacks as well as monitors the network. Both components report to the DIDS director, where the final analysis is done.

3.3 Summary

This chapter has demonstrated the diversity of possible attacks involved in computer systems. There are many countermeasures working at different network layers to protect networks against intrusions. As a second line of defence, an IDS plays an important role in computer security, especially in the fight against attacks launched inside networks. The two principal classifications of IDSs have been discussed. Signature-based IDSs focus on known attacks and vulnerabilities, and anomaly-based IDSs work alone with a reference model. Host-based IDSs collect local data, and network-based IDSs monitor network traffic through audit data. In the next chapter, we present a critical survey on current IDSs and discuss their specific limitations against the requirements of ubiquitous computing.

CHAPTER FOUR

LIMITATIONS OF CURRENT IDSS FOR UBIQUITOUS COMPUTING

Traditional IDSs were developed for wired networks. To our knowledge, there is no IDS yet, which has been particularly proposed to meet the special requirements of ubiquitous computing. However, with the continuous development of IDSs, especially the progresses made on wireless ad hoc networks and distributed IDSs, we believe that some existing solutions could be extended for intrusion detection in ubiquitous computing. In this chapter, we give a critical review of existing solutions that have been utilized in intrusion detection. Their benefits and limitations are discussed.

4.1 Cost-efficient solutions

As we discussed earlier, many appliances in ubiquitous computing have limited resources. For those battery-powered devices, an apparent issue is how to implement an IDS efficiently. A natural idea is to disable the IDS when it is not needed. This is not as straightforward as it sounds since one of the most desired features for an IDS is real-time detection. An improper deactivation of the IDS might be exploited by attackers and cause severe consequences.

Yi-an Huang proposed a cooperative IDS for ad hoc networks [68]. He assumes the hosts are organised into clusters. The nodes in the same cluster periodically choose a cluster head as an agent to execute intrusion detection tasks. Since only the cluster head needs to implement the detection module, this method alleviates the overall CPU usage and network overhead compared with running

the IDS at each individual node. All the nodes in a cluster have the possibility to be chosen as the cluster head, so it is necessary to pre-install the IDS modules on them. This method requests the nodes in an ad hoc network are powerful enough to carry out the IDS tasks independently. It is difficult to meet this requirement in ubiquitous computing due to different resource characteristics. However, the idea of a cluster-based IDS did inspire us during the initial design of SUIDS.

Wenke Lee and his colleagues tried to reduce IDS costs from another aspect [46, 94]. They divide the costs of an IDS into three parts: operation cost, response cost and damage cost. The damage cost is quantified based on an intrusion's type and its target. A multiple model cost-sensitive machine learning technique is proposed to produce models that are optimized for user-defined cost metrics. In other words, the optimized model reduces the detection costs by intelligently rearranging detection rules. The fundamental is that an IDS's operation cost is proportional to the number of rules that have been examined. A similar technique was also used in the paper [108]. It suggests improving the IDS performance by ranking and selecting detection features according to their criticality. All these works try to reduce the detection costs from the inside mechanism of an IDS. However, quantifying the IDS cost is a complicated and costly work. In different scenarios the same attack may cause unequal losses. The main problem is the lack of a common standard.

Most network-based IDSs identify intrusions by packet analysis. With the continuous growth of network infrastructures, network traffic has exploded during the past decade. To achieve real-time intrusion detection and reaction, the network-based IDSs need more efficient strategies. The paper [93] copes with the IDS overload problem by running performance monitoring on each node. Similar to current QoS techniques used in network devices [114], this performance monitoring system puts the most crucial event in front of others. Available techniques include rule selection and scenario analysis (predicting the forthcoming attack). Although in some ways the performance of IDS has been improved, it cannot solve the overload problem thoroughly. That is one reason we think host-based IDSs are favoured in ubiquitous computing.

From the respect of intrusion reaction, an adaptive response strategy can reduce the overall IDS costs as well. Although directly disconnecting or shutting down a system being attacked can immediately prevent a threat from malicious

opponents, it could also be seen as a success of a denial-of-service attack. The paper [133] proposed an adaptive response mechanism by quantifying the IDS's parameters such as the false alarm rate, detection confidence and damage cost. The system will choose a suitable response strategy based on the calculations of these parameters. Once again, the procedure of quantification is too complicated and lacks a common standard.

4.2 Hierarchically distributed solutions

The Distributed Intrusion Detection System (DIDS) is the first distributed IDS [138]. It brought a new dimension to intrusion detection by facilitating the correlation and analysis of data from multiple sources. In DIDS the monitoring and analysis functions are distributed among several components. These components are a central manager, a single host monitor per host, and a single LAN monitor for each broadcast LAN segment in the monitored network. The host and LAN monitors are primarily responsible for detecting single events and known attack signatures which are relevant to the security of an individual system. The central manager has access to the distributed audit data gathered by the various monitors. It is responsible for analyzing and correlating the events reported by the host and LAN monitors. It is worth noticing that DIDS can potentially protect the hosts without monitors since the LAN monitor can report on the network activities of them. The LAN monitor checks traffics on its LAN segment and creates a profile. In particular, it audits host-host connections, services used and the volume of traffics. This is an important feature because in ubiquitous computing some nodes may not be able to afford a host monitor but will still need IDS coverage.

However, due to the following reasons the DIDS cannot be applied to ubiquitous computing directly:

- In ubiquitous computing the network infrastructures are heterogeneous. It simply may not be IP-based. We have to redeploy the LAN monitors to cover the entire network.
- DIDS requires that all hosts run C2 or higher rated computer systems [153]. The systems must have certain level of access control ability, so

users are individually accountable for their activities through their login sessions. In ubiquitous computing devices are heterogeneous and some of them may not fulfil this requirement.

- DIDS still relies on a central manager. It may not scale well in a ubiquitous network with tens of thousands active nodes.

The Graph Based Intrusion Detection System (GrIDS) is concerned with detecting intrusions such as worms that involve connections between many nodes [145]. It builds activity graphs to represent host activities in a network. The system being observed is divided into domains. In the graphs nodes represent network domains and edges represent network traffic between them. In GrIDS each domain builds its own graph and passes the graph and summary information up to its parent domain. This pattern makes the system scale better. Obviously, as the information passes up the hierarchy, the graphs become coarser. GrIDS uses a rule set to determine how to construct the graphs based on incoming and previous information. Rules are also applied to decide whether or not a graph is suspicious.

The main problem of GrIDS for ubiquitous computing is that GrIDS was designed for static wired networks. It is not suitable for topology-varying environments. In ubiquitous computing many nodes are featured with mobility. It is difficult to construct an activity graph for such a large-scale network with dynamic topology.

EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) is also proposed to monitor distributed networks [115]. As concluded in [82], EMERALD uses a three-layered hierarchy to realize intrusion detection in a large-scale system. Each of the three layers consists of monitors. Each monitor may have its own anomaly and misuse detectors. The layers are named as: service (lowest), domain-wide, and enterprise-wide (highest). The service layer monitors a single domain. The monitors at the domain-wide layer accept input from the service layer and attempt to detect intrusions across multiple service domains. Likewise, the enterprise-wide monitors accept input from the domain-wide layer and attempt to detect intrusions that cross the entire system. Monitors may subscribe to information from other monitors at the same level and lower.

There are some limitations about EMERALD:

- The cooperation between monitors at the same layer is achieved by

subscriptions. This mechanism will introduce additional overload into the network. In ubiquitous computing system resources are crucial.

- The aim of EMERALD is to detect inter-domain attacks. It is not enough to have only EMERALD as an IDS in a ubiquitous computing environment. It has to collaborate with other host-based IDSs to provide integrated protection.
- The separation of domains is a remaining issue in a topology-varying environment like ubiquitous computing. In some cases it is hard to maintain the hierarchy of monitors.

4.3 Cooperative solutions

The IDSs mentioned above all have a hierarchical architecture. This architecture relies on a central controller. In a ubiquitous computing environment sometimes it may not have such a central point. This issue has already been noticed in wireless ad-hoc networks.

The paper [160] proposed a cooperative model for an IDS in a mobile ad-hoc network. In this model every node participates in intrusion detection and response. Each node has an IDS agent that is in charge of local data collection and local detection. A local detection engine runs independently. When an anomaly is detected in local data or there is inconclusive evidence, neighbouring IDS agents will be required to participate in global detection by using a cooperative detection engine. A local response module triggers local actions, e.g. alerting the local user, while the global module coordinates actions among neighbouring nodes, e.g. determining a remedial action. A secure communication module provides a high-confidence communication channel among IDS agents. The components are structured as shown in Figure 4.1.

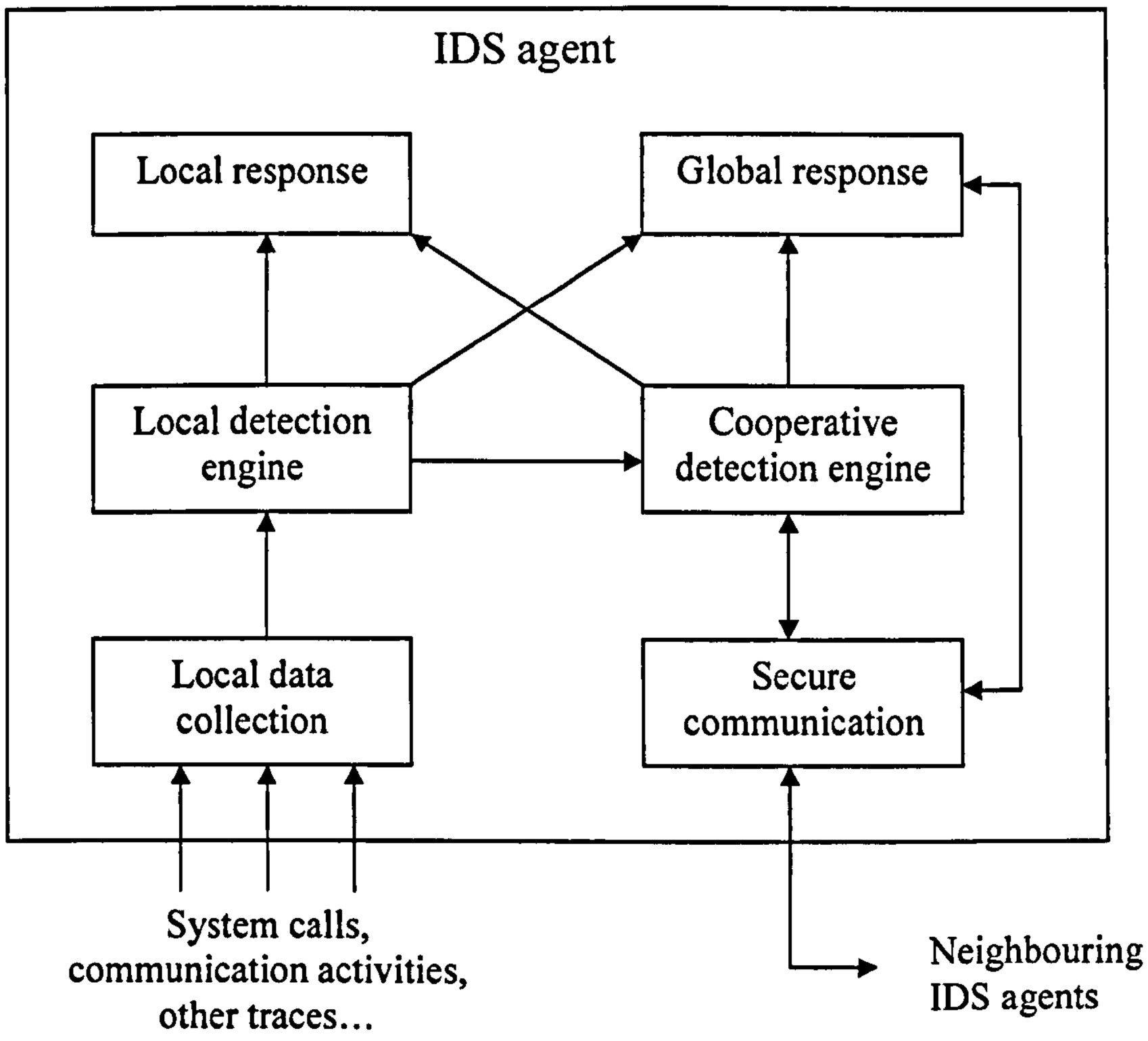

Fig. 4.1 A conceptual model for an IDS Agent.

The main contribution of this work is that it presents a distributed and cooperative intrusion detection architecture. The design of actual detection techniques, their performance as well as verification, however, were not addressed in the paper. Similar to paper [41], although their architectures are fully distributed, applying them to ubiquitous computing still requires further research. It is because in ubiquitous computing not all the nodes are guaranteed the ability to implement a local IDS agent independently. This model does not consider how to protect those incapable nodes.

Indra is a distributed scheme based on sharing information between trusted peers in a network [75]. It guards the network as a whole against intrusions. Indra brings a proactive and P2P approach to intrusion detection. The goal of Indra is to distribute intrusion attempt information (gathered by an intended victim) among all interested peers in a P2P network. Each interested peer runs a special Indra daemon. It watches out for intrusion attempts and also enforces access control based on its memory of earlier attempts. The chance that at least one of the peers

does notice an attack to which it is not itself vulnerable can be improved in relation to the number of peers, the heterogeneity of the peers (operating systems and/or applications), and the currency level of the applied security patches. The P2P network has to be reliable and trustful. This is achieved by applying a trust management scheme like the Web of Trust as known from PGP [144].

Indra is independent of a central controller. Because the ubiquitous computing is a heterogeneous environment, the idea for at least one machine to find an intrusion attempt, to which it is not itself vulnerable, is quite attractive. However, Indra has certain level of requirements on individual nodes as well. Each node has to implement the Indra daemon independently. The management and communication overload introduced by the P2P scheme hinders its application.

4.4 Mobile agent based solutions

A mobile agent is a software entity, which is capable of continuously and autonomously moving throughout networks and intelligently implementing certain tasks. The development of distributed IDSs and software agents has led to the idea of using mobile agents in intrusion detection. Mobile agents offer several advantages when applied to IDSs [77]. These advantages include:

- Reducing Network Load - Mobile agents can carry about intrusion detection algorithms with themselves. This mechanism can avoid transferring huge amounts of data (e.g. audit files) to data processing points.
- Overcoming Network Latency – Since mobile agents can operate locally on compromised hosts, they can react faster than the detection modules coordinated by a central point in a hierarchical IDS.
- Autonomous Execution – Because mobile agents are independent units, they can operate offline and autonomously. Even if portions of the IDS get destroyed or separated, the mobile agents will remain functional. This feature increases the fault tolerance of the overall system.
- Platform Independence - The agent platform allows agents to travel in a heterogeneous environment and inserts an OS independent layer between

the hosts and the IDS agents. This is important in ubiquitous computing.

- Dynamic Adaptation - The mobility of agents can be used to reconfigure the system at run-time by letting special agents move to a location where an attack currently takes place to collect additional data.
- Static Adaptation (Upgradeability) - It is important for an IDS, especially a misuse-based IDS, to update its attack signature database and detection algorithms properly. It is simpler to write updated agents and send them on duty while the IDS keeps running when new signatures are available.
- Scalability – Using distributed mobile agents can divide the computational load between different machines and reduce the network load. This enhances the IDS scalability and additionally supports fault-resistant behaviour.

The biggest concern over mobile agents is the security of themselves. Researchers are still working on protecting mobile agents against malicious hosts [44, 64, 130]. However, due to their unique advantages remarked above, we still see an increasing trend of applying mobile agents into IDSs.

Autonomous Agents For Intrusion Detection (AAFID) is featured by autonomous agents [140]. An AAFID system can be distributed over any number of hosts in a network. Each host contains a transceiver, filter (optional) and any number of agents. Agents implement specific functions and monitor interesting events happening at the host. They may use filters to obtain data in a system-independent manner. The agents cooperate in a client-server fashion by sending their findings to the transceiver where it is further processed. A transceiver is a per-host entity that oversees the operation of all the agents running on the same host. It has the ability to start, stop and send configuration commands to the agents. The transceivers report their analysis results to one or more monitors. Each monitor oversees the operation of several transceivers. Monitors have access to network-wide data, and therefore they are able to perform higher-level correlation and detect intrusions that involve several hosts. Eventually, a particular monitor is responsible for providing information and getting control commands from a user interface.

Comparing with DIDS, GrIDS, and EMERAND, AAFID is more flexible and adaptive due to the usage of autonomous agents. But it still relies on a central entity (monitor) where events are collected and related. Moreover, AAFID did not

consider how to cope with the heterogeneous issue of ubiquitous computing.

Sparta is another mobile agent based IDS [89]. In Sparta a single event is described by specifying appropriate values for its attributes. A number of events can be connected by defining temporal or spatial relationships between them or imposing certain constraints on their attributes (creating patterns). Interesting events are locally collected and stored. The collection of all local information can be considered as a distributed database. A user may issue queries in an Event Query Language (EQL) to search for a set of events that fulfil his/her desired constraints [29]. The mobile agents in Sparta can correlate distributed events and deduce knowledge from different hosts in a fully decentralized manner. It starts its task by contacting the directory service to obtain a list of hosts that match the constraints given in the FROM clause. These hosts are then visited in arbitrary order. Eventually the mobile agents will return results to the user.

The implementation of Sparta is very complicated. Its special requirements on the directory service and event query language constrain its application to a large-scale network. In ubiquitous computing nodes are free to join or leave the network. A directory service can hardly guarantee network updates in due course. Furthermore, functional mobile agents in Sparta put even higher requirements on individual nodes.

Except for the two systems mentioned above, other researches utilize mobile agents in a quite similar way. We deem them as a function-based solution because in these systems mobile agents are classified based on their particular functions. Normally such an IDS includes surveillance agents (data collection agents), decision-making agents and response agents. Each type of agent implements specific functions. In paper [83] the author distributes these agents into a cluster-based ad-hoc network. An elected cluster head is in charge of monitoring the network traffic within its cluster and making decisions. In paper [107] the surveillance agents roam around to find any suspecting event. If any anomaly is detected in the network, the surveillance agents will call for further checks. Because there are hundreds of identified attacks and system vulnerabilities, a mobile agent cannot be sensitive to all of them. The papers [8, 61, 87, 101, 164] use lightweight mobile agents to overcome this issue. In their systems the detection work is distributed by asking each agent to prevent one particular threat only. A central manager dispatches different kinds of agents into the network. If

an alarm is raised, the manager will inject more related agents into the network to further help. The difficulty is to effectively distribute those agents in relation to when, where and what kind of intrusion will take place.

Basically, function-based solutions also need a central controller. It provides the services like agent initialization, agent distribution, and user interfaces. These IDSs can effectively distribute the processing workload throughout the network being monitored. They have better adaptability and flexibility, though the reliance on the central controller may limit their scalability. Besides, like any other immature techniques, mobile agents introduce additional weaknesses especially in respect of trustworthiness.

4.5 Summary

We have summarized the aforementioned IDSs and compared them together in Table 4.1. The following points were boiled down to:

- Most existing IDSs did not consider resource constraints. The reason is that they were designed for wired networks or ad hoc networks (laptop or PDA based). The requirements for IDSs in these environments are not as strict as those in ubiquitous computing environments. The resource consumptions on IDSs need to be further reduced.
- An IDS in ubiquitous computing environments should be characterized by a distributed auditing scheme followed by distributed intrusion detection analysis. Conventional hierarchical architectures are not suitable due to the dynamic features of ubiquitous computing. Cooperative architectures have relatively higher resource requirements on individual nodes. A mobile agent based IDS is very promising, but a crucial issue needs to be considered - enhancing the security of the mobile agents to avoid introducing new flaws.
- Last but not the least, it is important to protect the nodes that lack abilities to implement an IDS module independently. Within our knowledge, only DIDS and GrIDS provide a solution for those incapable nodes. From this aspect, a network-based IDS is advantaged. A LAN or cluster manager is needed to take care of the incapable nodes. Remaining issues include what

are appropriate audit data sources and how to cope with a heterogeneous environment.

In this chapter we presented the state of the art in intrusion detection related to ubiquitous computing. To realize ubiquitous computing, further efforts are still needed in the areas of both hardware and software. A carefully designed security scheme can ensure that all the work shall be done in the right way. Although the research in intrusion detection started decades ago, its application to ubiquitous computing is still fresh. As discussed earlier, existing solutions on resource-efficiency and system architectures cannot fulfil the special requirements of ubiquitous computing. Specifically, an IDS in ubiquitous computing should not require to transmit or process a large amount of audit data or attack signatures; a centralized detection scheme should be replaced by a distributed or cooperative system architecture; host-based and network-based approaches should work together to provide all-sided protection. In the next chapter we will introduce the framework of our proposed SUIDS based on this analysis.

Table 4.1 A summary of introduced IDSs.

Introduced IDSs	Resource efficiency	Independence of central point	Consideration of incapable nodes	System scalability	Overhead introduced
DIDS	No	No	Yes	Yes	Low
GrIDS	No	No	Yes	Yes	Low
EMERLAND	No	No	No	Yes	Low
IDSs for Ad Hoc	No	Yes	No	Yes	High
Indra	No	Yes	No	No	High
AAFID	No	No	No	Yes	Low
Sparta	No	No	No	No	High
Function based MA	Yes	No	No	Yes	Low

CHAPTER FIVE

SUIDS AND ITS SIMULATION

Most IDSs strive to be general purpose and able to detect attacks and anomalies in any environment. This is clearly a very hard challenge, and as we presented earlier, it causes problems in ubiquitous computing. In this chapter, we introduce the framework and simulation of our proposed SUIDS, an adaptive and resource-efficient intrusion detection system with a novel service-oriented auditing mechanism and flexible user-centric design. By working together with service-oriented agents, it can reliably and effectively detect malicious activities of inside users. SUIDS is suitable for heterogeneous environments such as ubiquitous networks. It has the following features: a reliable auditing mechanism, a resource-efficient intrusion detection scheme, and a flexible system architecture.

5.1 Design of SUIDS

5.1.1 Application scenario

Ubiquitous computing still is an ongoing research. As a prospective view of future direction, further efforts on both hardware and software are needed [102]. Although there are only few prototypes implemented in research labs [54, 142], we believe that a smart space is an appropriate case as our research scenario. Fig. 5.1 illustrates Mike's smart home in which two PCs on the network backbone are connected with a domain management node. Mike's PDA is equipped with wireless connection and able to operate some appliances such as an electric door lock and a smart refrigerator. He could open the home door by sending a login message to the door lock, or check the food information stored in the smart

refrigerator through his PDA. A wall screen is used to display any message, document, picture or video clips taken by a camcorder. A broadcast service point regularly sends him newsletters based on his subscription status. All these devices are seamlessly connected together through wired or wireless connections and provide services to Mike.

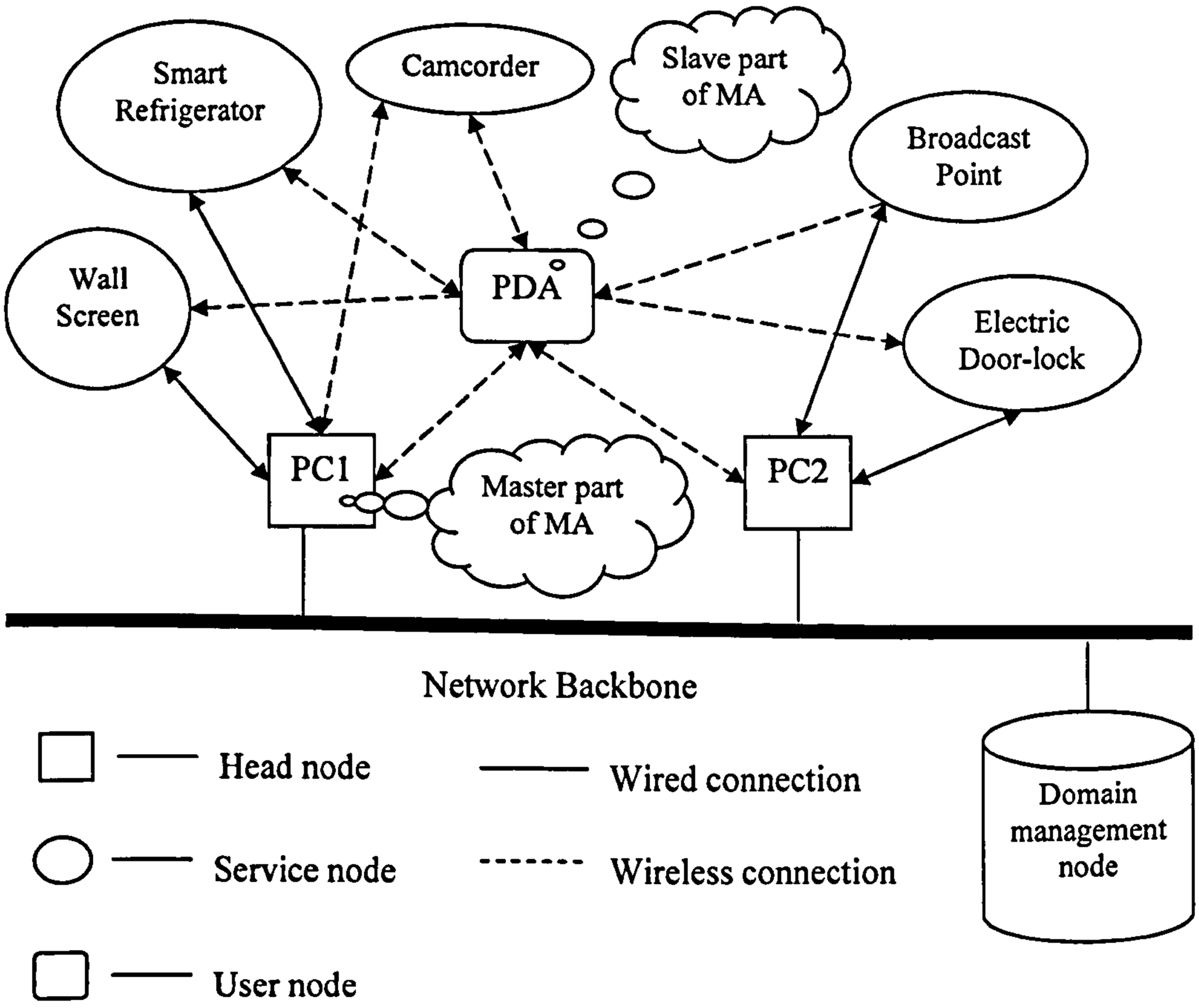

Fig. 5.1 Example of Mike's smart home.

In this case several attacks could take place against Mike's smart home:

- Confidentiality Attack: Unauthorised access to system resources and the information stored within these resources. Example: Mike's friend, Paul, uses a fraudulent ID to access Mike's folder on one of the PCs to gain some confidential information.
- Integrity Attack: Unauthorized modification to the state of the system and to the information stored within the system. Example: Paul alters the data about the food stored in the smart refrigerator. The modified information

may cause Mike unnecessary waste or threaten his health.

- Availability Attack: Unauthorized possession of the system resources in order to interfere with authorised users' normal access. It is well known as a Denial of Service (DoS) attack. Example: Paul tries to open Mike's home door by continuously sending login requests to the door-lock. Thus Mike cannot open it as the door-lock is always in a busy state or simply has turned off due to too many failed attempts.

5.1.2 Nodes classification

Nodes in Mike's smart home have diverse capabilities as they play different roles. For example, PCs are preferred for faster computing ability and higher network bandwidth in order to complete complex tasks in due course; camcorder and PDA emphasize smaller sizes to be easily carried about. Therefore we cannot treat all the nodes in the same way during the design of SUIDS. Before presenting the system architecture, some terms and a necessary classification of network nodes are explained first:

- *Domain management node*: Domain management nodes are in charge of the system management. They manage users' profiles and generate appropriate mobile agents for each user. Dividing the system into domains makes the system more scalable. There is one domain management node in each domain. Domain management nodes may cooperate together. For example, an individual company may form a single domain; the employees of the company register with the domain's management node; a visitor to the company needs to register with it first before he/she can use the system resources; the visited domain may contact the visitor's home domain to require necessary information such as the user's profiles.
- *Cluster*: A domain is composed of clusters. Each cluster has a PC-based central controller and all clusters are connected together. In a smart space a possible way of dividing clusters is based on rooms. This model is consistent with the viewpoint that ubiquitous computing is an evolutionary result of developing available techniques and integrating them together. Nowadays, one hardly finds an office with no PC in our department.

- *Head node*: Head nodes form a key part of the whole system. They are organised by clusters. There could be more than one head node in a cluster. The PCs in Mike's smart home are examples of head nodes. Head nodes are allocated on the network backbone with higher connection speeds and advanced operating systems. Head nodes take the most computing burdens of intrusion detection.
- *Service node*: Service nodes such as a smart refrigerator, camcorder and electric door lock are used to store information or provide specific services only. These nodes have very predictable running processes, open ports and traffic patterns. Sometimes service nodes are controlled by or provide services through head nodes. For example, a wall screen needs to get the content from a PC for the purpose of display.
- *User node*: User nodes are defined as those portable devices such as a user's PDA or smart phone. They have relatively powerful computing ability and advanced operating systems. Although user nodes usually were ruled out from intrusion detection, we have a different viewpoint since these devices start to offer application tools and are quickly becoming necessities in today's business environments [76]. In our design user nodes play an important role and they could be used to share the detection burden with the head nodes. It makes the system more scalable and resource-sensitive.

5.1.3 System architecture

SUIDS is a distributed application, dynamically deployed based on the classification of network nodes. The system is organized hierarchically with several tiers. Different tiers correspond to different network scopes that are monitored by intrusion detection modules.

For the case of the smart home shown in Fig. 5.1, SUIDS is divided into three tiers. Tier 3 consists of service nodes and user nodes. Detection modules running on tier 3 monitor system processes and network activities of these service nodes and user nodes. They generate service-oriented event records for the upper tier based on the monitored service usages and system operations. Tier 2 mainly consists of head nodes. Detection modules running on tier 2 analyze the event

records received from service nodes and user nodes that are within their corresponding clusters. They infer the status of the system, make decisions and take actions based on the collected event records. Tier 1 is in charge of the domain management. It holds users' profiles and generates appropriate mobile agents for

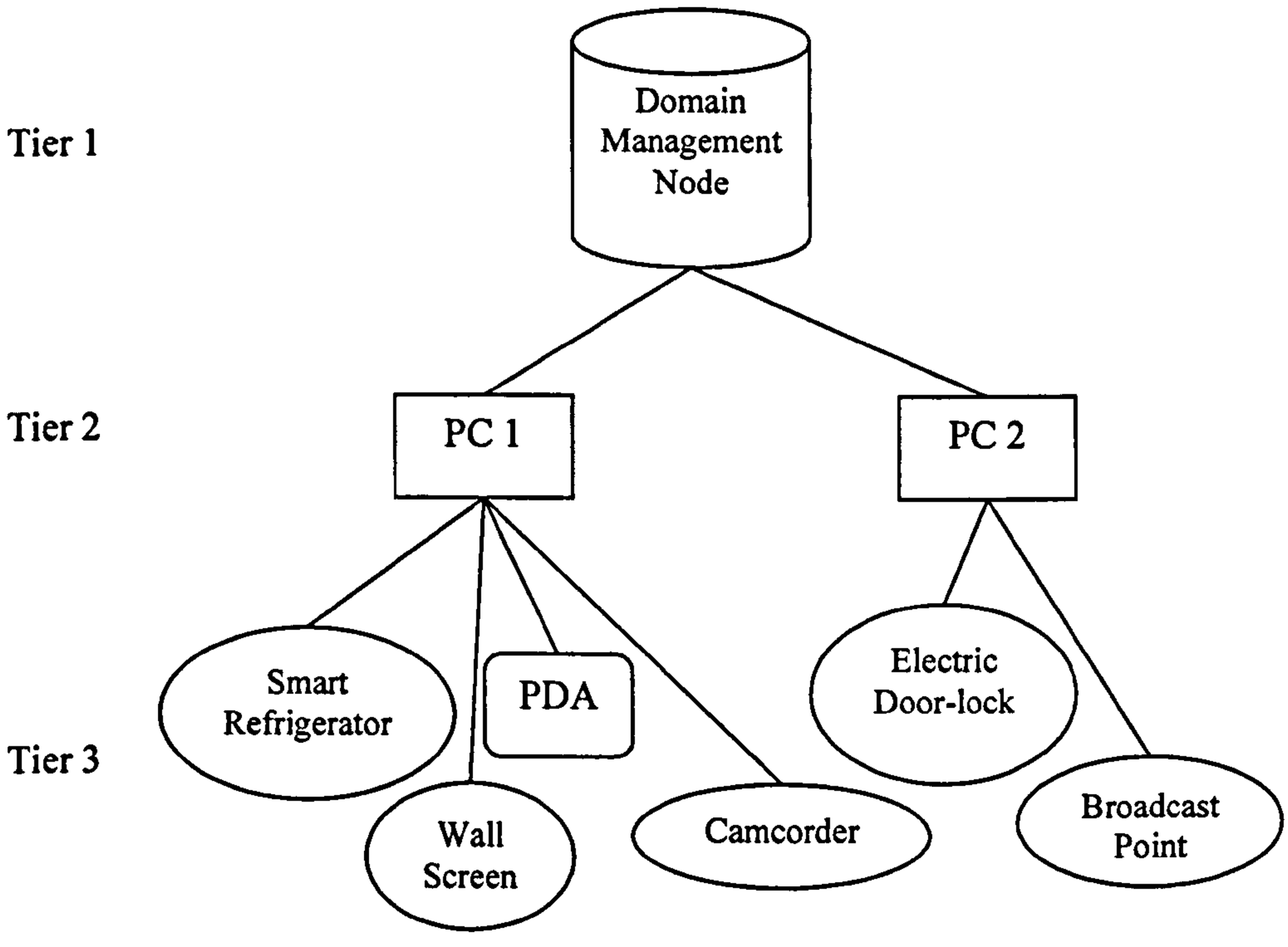

Fig. 5.2 System hierarchy for Mark's smart home.

each user. The system hierarchy for the smart home example is shown in Fig. 5.2. There are two small clusters on tier 2. The smart refrigerator and wall screen belong to PC1's cluster. The electric door-lock and broadcast point belong to PC2's cluster. The user's PDA and camcorder are portable devices. They are temporarily in cluster 1 and might move to cluster 2 later.

5.1.4 Service-oriented intrusion detection

Currently there are only a limited number of services offered publicly by computer networks, for example the HTTP, DNS and FTP services [88]. Normally these services are provided by specific network servers. However, with the trend of computerizing existing devices, the pattern of service providing will become highly distributed. More and more services will be available through the

networks and provided by specific devices (here referred as service nodes). Just like Mike's smart home, he can open his electric door by sending an order to the door-lock. In ubiquitous computing, users' activities carrying through computer networks will not only be limited to certain services, but extend to daily routines.

Our intention is to provide a distributed IDS for heterogeneous networked appliances that possibly have limited capacities. In current work, the protection of services relies on the powerful computing ability and enormous storage space of network servers for intrusion detection. For example, HIDSs require a server be able to monitor and store a wide range of audits, and NIDSs require the server be able to collect substantive network traffic. Unfortunately, these requirements are hardly fulfilled by some devices in ubiquitous networks. Although some researchers try to use lightweight mobile agents to ease the burden on a target system being protected, most of them are designed for homogeneous environments [41, 61]. Mobile agents in such an environment are normally signature-oriented, blindly trying to find specific flaws in any target system. A single device may have to execute tens or even more such lightweight agents in order to gain an all-sided protection. Obviously it is not an effective solution for ubiquitous computing.

SUIDS overcomes this issue by generating service-oriented mobile agents for each kind of devices. Integrating with service specific knowledge, it decreases the system complexity and makes it more practical and resource-efficient. In SUIDS, the service nodes are required to remember their corresponding head nodes and send event records to them during executions. To achieve it the service nodes need to register with head nodes first. This is a reasonable requirement for ubiquitous networks as the service nodes must let people notice their existences before providing services. The head nodes will ask the domain management node to send specific mobile agents for the service nodes within their clusters.

One obstacle confronting us is the high diversity of service nodes in terms of different functionalities and capacities on computing ability, storage space, communication bandwidth, and energy supply. If we look at today's market, different types of product are produced by hundreds of manufactures. In ubiquitous computing, these products have to seamlessly work together and provide services to users. Currently some research efforts focus on service broking in ubiquitous computing [50, 54]. The purpose is to establish a

mechanism to make the services provided by networked appliances (service nodes) effectively available to the users. We assume there is such a service broking mechanism semantically available to SUIDS, though the detail of service broking is beyond the scope of this thesis. Thus the domain management nodes can abstract necessary information about the service nodes, for example their OS and I/O mode, to generate appropriate service-oriented mobile agents. The mobile agents have better understanding of the service nodes on both hardware and software levels and are able to react correspondingly to each kind of them. The mobile agents should be platform-independent. Sun Java [78] is widely used in similar environments as a development tool.

5.1.5 Concept of a user-centric model

Based on each user's daily activities, SUIDS generates a service-oriented profile for the user. It is used to identify any abnormal usage of the services under a certain user ID. Notice again that the definition of 'service' in our system is not the same as the conventional applications offered by network servers. It could be any service provided by networked appliances.

We use another type of mobile agent to follow users around and connect them to the system's Tier 2 and 3. To distinguish these agents from those for service nodes, we call them *user agents* and *service agents*, respectively. Each user agent has two components: detection modules and user profiles. Service agents on Tier 3 (service nodes) collect information about a user's activities and send event records to the user agents. The user agents on Tier 2 (head nodes) analyse the records sent by the service agents and take the corresponding actions. In this way, the detection module is kept away from the audit module so as to increase the robustness of SUIDS and release burdens on the service nodes. Comparing with head nodes, service nodes are much constrained by their capacities such as power supplies. By allowing a designated user agent to follow a user, the SUIDS design can save lots of network resources since the user agent follows the user around and always tries to find the closest head node to the user. By giving mobility to the detection modules, SUIDS can achieve better performance in respect of resource-efficiency. The operation of the user agent and its data flows are shown in Fig. 5.3. The details of detection methods will be explained in the next chapter.

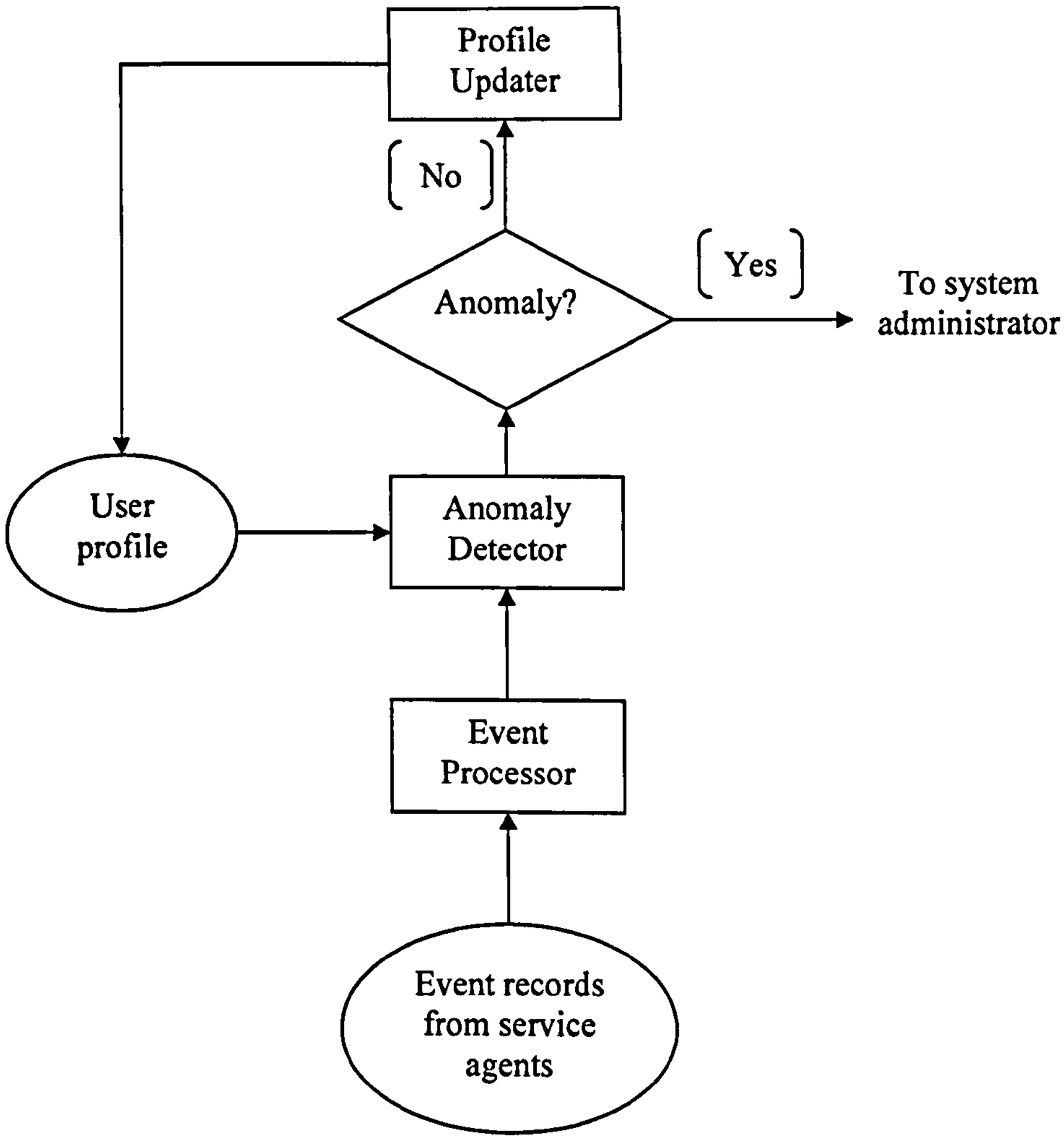

Fig. 5.3 High-level operation of the user agent.

In addition, the service nodes, which have received service requests from users, will dynamically form a defence wall against malicious inside users. As we discussed earlier, traditional NIDSs give an inside user opportunities to bypass the network intrusion detection. In SUIDS, service agents send event records to user agents according to their system states and real-time operations. This event-triggered design will let the user agents notice any user's activities within the networks. Therefore, intrusion detection in SUIDS is more reliable and the nodes outside the defence wall are beneficially released from burdensome intrusion detection surveillances. The service agents only need to monitor and record essential activities of users, depending on the system security policy. For example, if an authorized user asks for the information about the food stored in a

smart refrigerator, it may be allowed and not be recorded, depending on the given security policy; but if someone wants to login and change the system settings, it will be recorded and sent to the corresponding head node immediately.

The system structure of the proposed user-centric model is shown in Fig. 5.4. Comparing with existing solutions, the user-centric SUIDS has following advantages:

- In a ubiquitous network the number of users is much fewer than the number of service nodes. The user-centric model can remarkably decrease the system complexity in comparison with implementing the IDS on each node.
- Most services are requested by and then provided to users. The user-centric model can effectively collect network activities inside the ubiquitous networks.
- By separating the detection module from the audit module, risks stemming from the weak security features of mobile agents are reduced.
- A mobile user profile and detection agent can save more network resources.

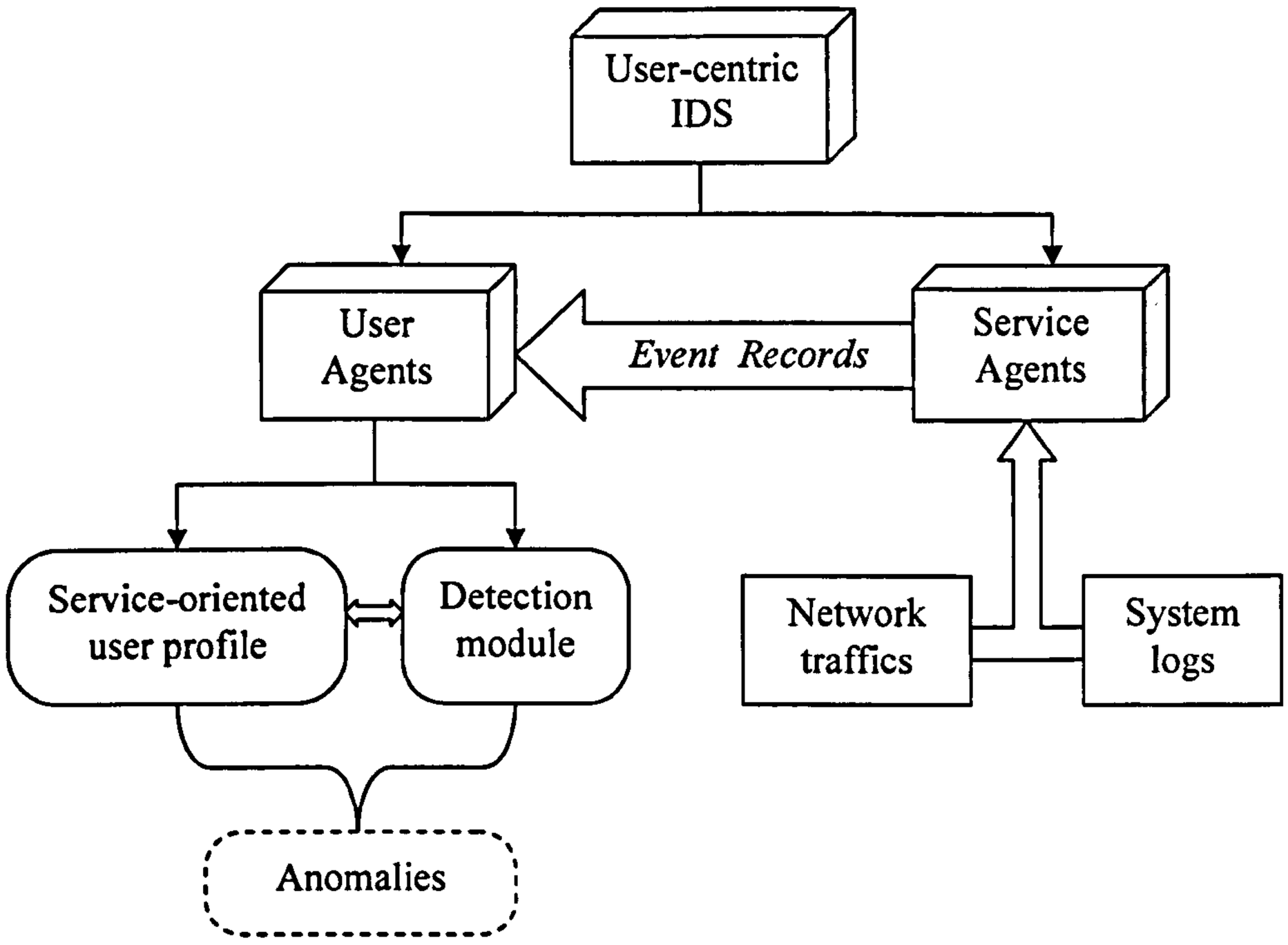

Fig. 5.4 System structure of the user-centric model.

5.2 Simulation of SUIDS

In order to test the feasibility and applicability of SUIDS, we created a simulation environment by using the Georgia Tech Network Simulator (GTNetS) [58]. GTNetS is a fully-featured network simulation environment that allows researchers in computer networks to study the behaviour of moderate to large scale networks under a variety of conditions. The programming language used is C++.

5.2.1 A simulation scenario

The first step is to define a proper simulation scenario. For research purposes, we first created a small smart home with fourteen nodes. Among these fourteen nodes, there are two head nodes, two user nodes, and ten service nodes. Fig. 5.5 shows the initial state of the simulated environment. The desktop icons represent

the head nodes; the PDA icons represent the user nodes; and the rest are service nodes.

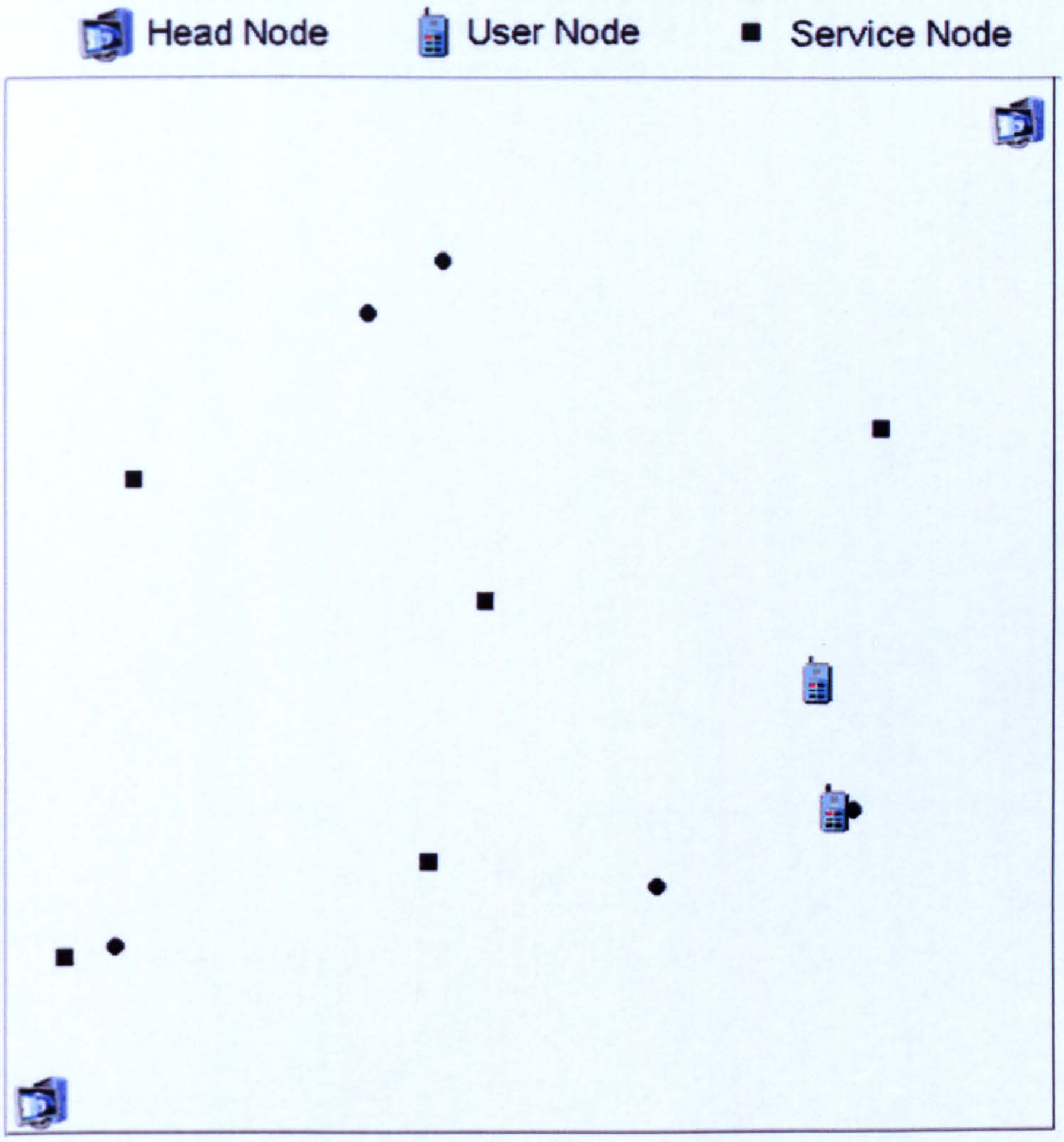

Fig. 5.5 Initial state of the simulated environment.

The user nodes in our experiments are mobile. Fig. 5.6 shows another snapshot of the simulation during its execution. Comparing with Fig. 5.5, we can see that the positions of the user nodes have clearly changed. The mobility pattern used here is the Random Waypoint (RWP) model [109], which is a commonly used synthetic model for mobility. Briefly, in the RWP model:

- Each node moves along a zigzag line from one waypoint to the next.
- The waypoints are uniformly distributed over the given area. In our simulation it is a 50 x 50 m^2 space.
- At the start of each leg a random velocity is drawn from the velocity distribution. In our simulation the human velocity is set to be uniformly

distributed between 0 ~ 2 meters/second.

- Optionally, the nodes may have so-called "thinking times" when they reach each waypoint before continuing on the next leg. The duration is also an independent and uniformly distributed variable. In our simulation it is between 0 ~ 10 seconds. Obviously, it is unlikely that people will continually move around within every 10 seconds. This assumption aims to reduce the simulation time.

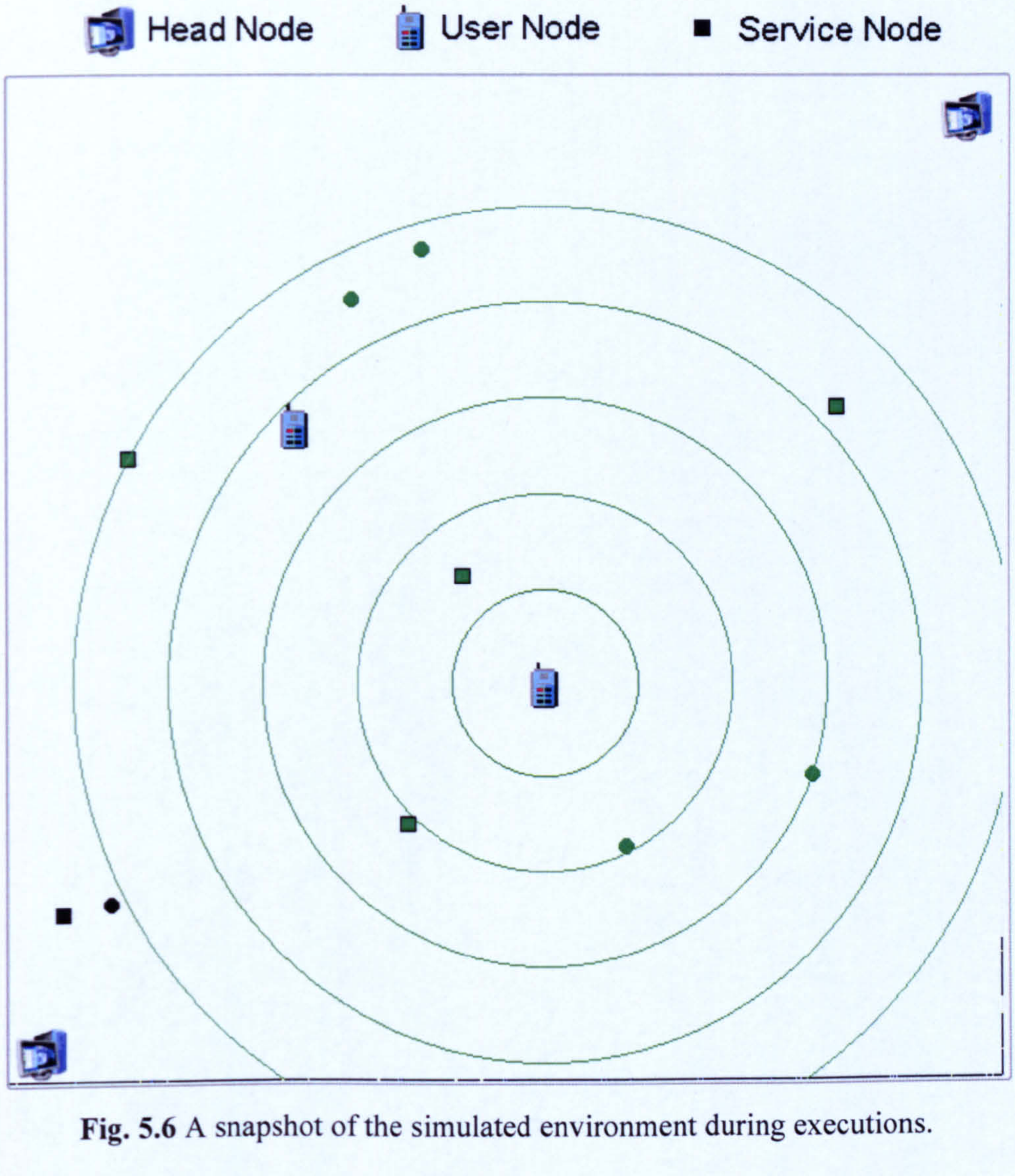

Fig. 5.6 A snapshot of the simulated environment during executions.

In current experiments we assume that all the nodes in our simulation are connected and communicate with each other through wireless connections, i.e. in an Ad Hoc pattern. The default routing protocol is DSR [81, 126].

5.2.2 Service nodes classification

Once a network topology is defined, the simulation must then introduce the flow of data through the defined network topology. In GTNetS, it is done by creating applications at various nodes, which in turn generate data demands on the network based on the application behaviour. In our simulation these applications are equal to the services provided by the service nodes. It is easy to understand that most network traffic is introduced by users' requests for services and the corresponding responses.

Instead of defining and creating service nodes one by one, we categorize them into several types based on their functionalities and characteristics. This classification could be extended in the future.

- The first type is the service nodes with only an on-off operation, e.g. a room light or an electric door lock. This kind of node receives orders from a user and executes the corresponding operations. In SUIDS we take the operation time, duration and frequency as auditing data. An authentication process might be involved before each session of data transmission.
- The second type is the service nodes with an on-off operation and an adjustable parameter, e.g. an adjustable central heating, a temperature-keeping kettle or a smart refrigerator. Authorized users are allowed to adjust the parameter as they want. In this case we take the operation time/duration and the value of the parameter as auditing data. If a device has more than one parameter, we only take the most security-related one into account in the current simulation. Our work can be extended to consider multitude parameters in the future.
- The third type of service node has an on-off operation together with a display function, e.g. a wall screen or a TV monitor. The definition of 'display' here has a wider meaning. For example, a printer will also be classified into this category since we consider the operation 'print' as a kind of 'display'. This type of node is used to 'display' images, documents or video clips. They are quite similar to the second type of service node discussed above. The difference is that one parameter is not enough to determine the effect of the operation. For example, printing high quality images requires a very different amount of system resources

from printing the same number of pages for a text file. Therefore, we use both the amount of received data and the number of displayed/printed pages as auditing data.

- The last type of service node is based on a client-server enquiry pattern. This pattern includes various possible operations. For example, a smart refrigerator may provide an enquiry service to allow a user to ask for the information about the stored food. And a camera may also provide a picture downloading service to the user. In this study we only simulate the two most common operations: Downloading and Enquiry. For both of them we monitor the operation time, duration and the amount of traffic in both directions as auditing data.

Most service nodes have a predictable traffic pattern and running process. The traffic patterns depend on both functions of the service nodes and behaviours of the users. For example, a broadcast service may either send updates regularly or be triggered by a request from a user. There are several types of traffic generators supported by GTNetS:

- CBR Application: An application that generates constant bit rate data between two nodes.
- On-Off Application: An application that offers an on-off operation. The durations of the on and off states are exponentially distributed.
- TCP Server: A simple model of a request/response TCP server. The server binds to a specified port, and listens for connection requests from TCP peers. The data received from the peers specifies how much data to send or receive.

5.2.3 User scenario

Normally services are triggered by users. A usage pattern, including the time, duration and frequency, depends on its user's behaviour. The user's behaviour is a dynamic result and hard to be simulated. We defined two simplified scenarios for the simulation of normal usage patterns.

- **Random choosing scenario:** There are three random variables in this scenario: a random node, a random busy duration and a random idle duration. In this scenario a user will randomly pick a service node to

activate its service (start transmitting data) for a random duration. Then the user will turn into idle for another random duration. We assume a user will only communicate with one service node at any given time.

- **Nearest choosing scenario:** In this scenario a user will choose the nearest service node to communicate. Comparing with the random choosing scenario, the nearest choosing scenario does not need the variable of a random node anymore. Which service node will be activated depends on how far it is away from the user. Actually it is related to the user's mobility pattern.

5.2.4 Intrusion auditing

With the assistance of the GTNetS tracing ability, we can get the system's trace file. In this trace file all packets are recorded. In addition, we embedded a recording function in each service node. The purpose is to generate event records according to node activities. We selected three of the most common activities for our research. Actually, no matter what kind of service the service nodes provide, we can always abstract three basic operations from a user's point of view: PROBE, GET, and SET. PROBE represents the process of communication negotiation and the setting up of connections. PROBE could be used as the first step of an attack. GET represents the process of receiving data from service nodes. For service nodes GET means the operation of 'data out'. GET could cause a denial of service attack since the unauthorised possession of the system resources might interfere with legitimate users' normal access. SET represents the process of sending data to or configuring the service nodes. For the service nodes SET means the operation of 'data in'. SET could change the status of some service nodes and take them into unwanted conditions.

In the collected audit data, the corresponding time (including the occurrence time and duration), involved parameters, and amount of data (including both directions) of each operation are recorded by SUIDS. The final step is to investigate the collected trace files and event records, analyze the service usages and create user profiles. This is done by using the Java programming language, to enable its migration from the simulation to the real world. The detection methods and simulation results are reported in the next few chapters.

5.3 Summary

In this chapter we focused on the new system architecture and detection mechanism of SUIDS. The simulation of SUIDS is made by using GTNetS. In conclusion, we list the key novelties of SUIDS:

- Portability and Reliability: The usage of portable user profiles makes SUIDS able to detect intrusions without a priori knowledge of security flaws in a target system. This mechanism is highly portable and consistent with the mobility feature of ubiquitous networks. The service-oriented detection method protects service nodes from malicious inside users. Service nodes spontaneously form a defence wall against a malicious user. Any activities carried by the user through the network will be recorded and analyzed by head nodes afterwards.
- Heterogeneity: There are thousands of diverse devices in ubiquitous computing environments. The classification of network nodes could help us effectively organise the entire system. Moreover, by cooperating with a service broking system, SUIDS will generate service-oriented mobile agents for specific devices. The service agents have better understanding of the service nodes on both software and hardware levels, and thus overcome the blindly roving problem caused by the homogeneous design of other mobile agent based IDSs.
- Resource-efficiency: Resources are crucial in ubiquitous networks, especially for those resource limited service nodes. In SUIDS, the classification of network nodes helps to effectively balance intrusion detection and resource consumption. The novel user-centric design can reduce the extra resource consumptions of intrusion detection on communications. More efforts had made on it and will be explained further in chapters eight and nine.

CHAPTER SIX

REAL-TIME INTRUSION DETECTION WITH A STRING-BASED APPROACH

SUIDS is designed for ubiquitous computing environments like a smart home/office. It adopts a novel auditing mechanism and flexible system architecture to meet the special requirements of ubiquitous networks. In this chapter we explain in detail about the detection methods used and experiments carried out during the implementation of SUIDS. Specifically, it shows how a string-based method is used in a user profile to represent the user's short-term behaviour in due course; and how an appropriate string length and threshold value are determined in order to balance the system's false alarm rate and detection effectiveness. As a result, SUIDS achieves real-time intrusion detection in ubiquitous networks with a lightweight and adaptable detection model.

6.1 Detection methods

6.1.1 Structure of event record

As a research scenario to demonstrate the design, we assume Mike lives in a smart home. He uses his PDA to open the home door, adjusts the temperature of the home central heating, and send documents to a printer in the house. In SUIDS, all these tasks carried out by Mike or on behalf of him will be recorded and connected to his account. For example, when Mike comes into a room A, two event records will be sent to the corresponding head node:

{Mike, Door_Room_A, open, 07:28:35am}

{Mike, Door_Room_A, close, 07:28:42am}

The head node may calculate the duration of this operation by using the time item in these records.

If a device has other security-related parameters, they will also be recorded. For example, when Mike uses a printer, two event records will be generated:

{Mike, Printer, print, 16pages, 11:12:23am}

{Mike, Printer, logout, 3.4Mb, 11:13:45am}

Here '16pages' indicates that Mike has printed 16 pages of documents in this session and '3.4Mb' indicates the amount of data that has been transferred during the same session. They are all security-related parameters. For example, a burst of requests on the printing service may indicate a denial-of-service attack or waste of the system resources. And it will be reflected by either a large number of print pages or a large amount of data transmission (e.g. printing one page of a high resolution picture may have the same effect as printing multi-pages of a text file). Eventually, these event records will be used to create Mike's profile.

6.1.2 Mathematical model

We use the statistical component of SRI's NIDES as our mathematical model [99, 162]. Instead of simply measuring the means or variances of variables, NIDES developed a more sophisticated statistical algorithm by using an X^2-like test to measure the similarity between short-term and long-term profiles. In NIDES, user profiles are represented by a number of probability density functions (PDFs). Assume S is the sample space of a random variable and events E_1, E_2, ..., E_k are a mutually exclusive partition of S. Let P_i denote the expected probability of the occurrence of the event E_i, P_i' denote the actual probability of the occurrence of E_i during a given time interval. The similarity between the expected and actual distributions is determined by the statistics:

$$Q = \sum_{i=1}^{k} \frac{(P_i' - P_i)^2}{P_i} \qquad (6\text{-}1)$$

If the cumulated value of Q exceeds a pre-determined threshold during a given time interval, an alarm will be raised. To utilize this statistical component we have to define a new model to specify the service-related factors, which can effectively represent a user's both long-term and short-term behaviours.

6.1.3 Historical statistics: representation of long-term behaviours

In SUIDS, a user's long-term behaviour is represented by probability distributions. They indicate the possible results and corresponding probabilities of a user's each kind of action. For example, statistical results may suggest that the typical time for Mike to open his home door during a day is between 8-9am and 5-6pm. This action rarely happens during other time. Thus we can get the following probability distributions for the action of opening the home door:

{Door, open, 1-2, 3%, 8-9, 48%, 17-18, 45%, 22-23, 4%}

Where '1-2' represents the door opening time, i.e. between 1-2am; and '3%' represents the statistical probability for opening the door during this period.

Similarly, we can also represent Mike's behaviour regarding his usage of a printer. Assume the recorded largest number of pages Mike had ever printed in one transaction is 200. Thus we can divide it into 10 possible groups: 1-20, 21-40, ..., 181-200. The occurrence probability for each group is:

$$P_i = \frac{E_i}{E} \tag{6-2}$$

Where E is the total number of records, E_i is the number of records for the i^{th} group.

Assume the expected probability distributions in turn are 38%, 36%, 20%, 1%, 0%, 0%, 3%, 0%, 1%, 1%. If Mike prints 30 pages in the current transaction, the partial similarity factor Q_2 is:

$$Q_2 = \frac{(P_2' - 36\%)^2}{36\%} \tag{6-3}$$

Where P_2' denotes the actual occurrence probability of E_2 (i.e. printing 21-40 pages) during a given time interval.

Except for the printed page number, other parameters such as the amount of data transferred and processing time occupied by each session are also monitored and taken into account in a similar way.

6.1.4 String: representation of short-term behaviours

The remaining problem now is to get the value of each actual probability P_i'. Some IDSs use a time interval to determine a detection window, i.e. each event only makes effect during a certain period. Because SUIDS is a distributed and

mobile system, the time-based detection window will introduce the synchronization issue and make the system more complicated.

Thus in SUIDS we proposed a string-based method to determine the detection window. The 'string' is used to indicate a user's short-term behaviour. For example, if the last 100 printing operations can effectively represent Mike's short-term behaviour regarding his usage of the printer, a string with the length of 100 will be set to follow the printing probability distributions in his profile. Each character of the string represents one of his historical printing operations. The format of his profile becomes:

{Printer, print, 1-20, 38%; 21-40, 36%; ...;181-200, 1%. 19082031012...15001}

10 pairs 100 digits

The last item here records Mike's last 100 printing operations. We use number 0-9 to represent the 10 groups, i.e. number 0 indicates printing 1-20 pages, number 1 indicates printing 21-40 pages and so on. Every time when a new record comes, the earliest record will be discarded. The value of P_i' can be calculated immediately from this string by applying the following equation:

$$P_i' = \frac{E_i'}{L} \tag{6-4}$$

where E_i' is the number of occurrence of the i^{th} group in the string, and L is the length of the string.

The length of the string is variable. It depends on the system's requirement and characteristics of each event. As will be explained in the next section, longer strings may decrease the false positive rate, but at the same time the false negative rate will be increased and more system resources will be used.

6.2 Experiments and results

As we explained earlier in chapter five, we created a simulation environment by using the Georgia Tech Network Simulator (GTNetS) [58]. All the nodes in our simulations are connected and communicate with each other through wireless connections in an Ad Hoc pattern. The default routing protocol is DSR [81, 126]. Fig. 5.6 shows the snapshot of the simulated environment. User nodes in our

experiments are mobile. The mobility pattern is based on the Random Waypoint (RWP) model [109]. Several types of service nodes were also specified according to their traffic patterns and parameter characteristics.

The first experiment we carried out is to examine the false positive rate of SUIDS and see how the string length affects it. We set the string length from 10 to 100, respectively, and divide the audit data into two parts. The first half is used to create a user profile and the second half is used to test. Because the audit data is generated and collected under a normal circumstance, any alarm raised during this test will be considered as a false alarm. To get a low false alarm rate, the value of Q needs to be small. To investigate each factor's exact influence on Q, we only take the processing time into account at this stage.

Table 6.1 shows the increment of Q after loading the test data into the system, with a different string length. We can see that the increment of Q decreases as the string length increases. As expected, it indicates that a longer string is more accurate to represent the user's short-term behaviour. However, because the longer string also uses more system resources, we chose the length of 80 as our investigation sample. Actually other parameters such as a threshold value also play important roles in the determination of the false alarm rate.

Table 6.1 Increment of Q decreases as the string length increases.

Length	Q
20	89.5758
40	42.6789
60	28.7096
80	19.1924
100	13.4959

We use a set of threshold values from 0.5 to 3.0 to calculate the system's false alarm rate. Once the cumulated value of Q exceeds a predefined threshold, an alarm will be raised and Q will be set back to zero. The false positive rate is calculated by:

$$R_{fp} = \frac{N_a}{N_e} \tag{6-5}$$

where R_{fp} is the false positive rate, N_a is the number of false alarms that have been raised, and N_e is the total number of events that have been checked. There are total 854 event records in the testing data set. The results are listed in Table 6.2.

Table 6.2 False positive rate (*String length = 80, N_e = 854).*

Threshold	N_a	R_{fp}
0.5	32	3.75%
1.0	18	2.11%
1.5	12	1.05%
2.0	9	1.41%
2.5	7	0.82%
3.0	6	0.70%

As we can see, the false positive rate of SUIDS is quite low. A bigger threshold value shows a less 'sensitiveness' to the deviations from the user's long-term behaviour. However, we cannot use Table 6.2 to decide an appropriate threshold value yet as it is also related to the next experiment.

The second experiment is to examine the system's effectiveness on detecting anomalies. We generated another set of audit data. This set of data introduces anomalies or attacks by extending the processing time beyond the normal extent. The effectiveness of the system is represented by a hit rate. If an alarm is raised in connection with an event record, this record is regarded as being 'hit'. A high hit rate on anomalous event records is preferred. The equation to calculate the hit rate is:

$$R_h = \frac{N_a'}{N_e'} \tag{6-6}$$

where N_a' is the number of genuine alarms and N_e' is the number of malicious events. There are total 181 anomalous records in this data set. Table 6.3 shows the experiments results.

Table 6.3 Hit rate (*String length = 80, N_e' = 181).*

Threshold	N_a'	R_h
0.5	172	95.03%
1.0	165	91.16%
1.5	159	87.85%
2.0	157	86.74%
2.5	151	83.43%
3.0	149	82.32%

In most cases, the hit rate must be kept as high as possible since any ignored attack may cause serious damages to the entire system. A tolerable false alarm rate depends on the system's security requirements/policies. Normally, it is acceptable to have a false alarm rate less than 5%. So combining Tables 6.2 and 6.3, we think in this case when the threshold value is set to 0.5, SUIDS can achieve the best performance regarding both measures.

6.3 Summary

SUIDS is proposed for ubiquitous computing environments. It takes the limited capability and high heterogeneity of service nodes and high mobility of user nodes into account. In this chapter, we introduced the detection details of SUIDS. It adopts a string-based method to represent a user's short-term behaviour in real-time. The experimental results show that with a carefully selected string length and threshold value, i.e. length = 80 and threshold = 0.5, SUIDS can achieve a hit rate of 95.03% with only a false alarm rate of 3.75%. The problem with the string-based method is that it may need more system resources if the length of strings is set too long or there are too many different types of events. Consequently the size of user profiles might be too large to be transferred frequently. In the next chapter, we will introduce a chi-square statistic test to further improve the performance of SUIDS.

CHAPTER SEVEN

IMPROVED CHI-SQUARE STATISTIC TEST

In chapter six, we presented a string-based approach for SUIDS to detect anomalies. In this chapter, we refine the detection method of SUIDS in order to improve its performance in terms of both detection effectiveness and efficiency. An exponentially weighted moving average (EWMA) technique is used to smooth out observation values for the variables being tracked. In this way, the observation reflects the 'most recent past' characteristics of the variables in an online fashion. The technique applies a smoothing constant to a user profile to represent the user's short-term behaviour in real-time. The deviations between a user's short-term and long-term behaviours are measured by using a chi-square statistic test. As a result, SUIDS can measure not only the probability distributions of variables, but also their occurrence patterns.

7.1 Detection methods

In the last chapter we introduced a string-based method to determine a detection window. A 'string' is used to indicate a user's short-term behaviour in an online fashion. For example, if the last 100 printing operations can effectively represent Mike's short-term behaviour regarding his usage of the printer, a string with the length of 100 will be set to follow the printing probability distributions in his profile. Each character of the string represents one of his historical operations. There are two problems with this string-based approach. Firstly, it might cost more system resources if the length of the string is set too long or there are too many different types of events. Consequently, the size of a user's profile might become too large to be transferred frequently. Secondly, it does not consider the

possible correlations between historical records. The most recent and past records are treated equally. Some hidden patterns regarding a user's behaviour might be carelessly ignored.

7.1.1 Chi-square statistic test

In the papers [158, 159], the authors use a multivariate distance test to determine anomalies. Let $M = \{M_1, M_2, ..., M_N\}$ denote a set of N measures from a process; $M(j) = \{ M_1(j), M_2(j), ..., M_N(j)\}$ denote the j^{th} observation of these N measures. The distance from an observation to the mean estimate of the multivariate normal distributions is measured based on a chi-square statistic test:

$$D = \sum_{i=1}^{N} \frac{(M_i(j) - \overline{M_i})^2}{\overline{M_i}} \tag{7-1}$$

where $\overline{M_i}$ is the expected value of the i^{th} variable. D is small if an observation of the variables is close to the expectation. An alarm will be raised if D exceeds a pre-determined threshold.

In its previous applications, the chi-square statistic test is used to measure the correlations between the commands at a sole host. It monitors and records the invocations of these commands. In contrast with the sole machine environment, there will be a large number of possible event types in a ubiquitous network. Examining the correlations between all these events will be a computation-exhausting and time-consuming task. Moreover, because most of the events contain one or more security related parameters, simply measuring the occurrences of these events will be insufficient to identify some intrusions. Hence, in this chapter we apply this chi-square statistic test within each type of event by analyzing their quantified parameters. It focuses on a user's behaviour on each specific service. In this way, the chi-square statistic test can be beneficially used in a distributed system like a ubiquitous network rather than a single host.

Additionally, to add a time characteristic into an observation, we use an exponentially weighted moving average (EWMA) technique [56] to smooth out an observation value for the variables being tracked. The observation thus reflects the 'most recent past' characteristics of the variables. Assume S is the sample space of a random variable X, and $X_1, X_2, ..., X_N$ are a mutually exclusive partition

of S. Every time when an observation of X arrives, a vector of $\{X_1(j), X_2(j), ..., X_N(j)\}$ will be generated as follows:

if the j^{th} observation of X falls into partition X_i

$$X_i(j) = \lambda \times 1 + (1-\lambda) \times X_i(j-1)$$

otherwise

$$X_i(j) = \lambda \times 0 + (1-\lambda) \times X_i(j-1) \quad (7\text{-}2)$$

where j is the index of the current observation, λ is a decay rate. The most recent observation, i.e. the j^{th} observation, receives a weight of λ; the $(j-1)^{th}$ observation receives a weight of $\lambda(1-\lambda)$; and the $(j-k)^{th}$ observation receives a weight of $\lambda(1-\lambda)^k$. In the next subsection we will give an example about how to use this EWMA technique and chi-square statistic test to create user profiles and detect anomalies.

7.1.2 Case study: monitor Mike's usage on a printer

We can monitor Mike's behaviour regarding his usage of a printer by using this chi-square statistic test. Assume that the recorded largest number of pages Mike had ever printed in one transaction is 200. Thus we can divide it into 10 possible groups: 1-20, 21-40, ..., 181-200, and use numbers 0-9 to represent these groups, i.e. number 0 indicates printing 1-20 pages, as described in section 6.1.4. We initialize $X_i(0)$ to 0 for i = 0, 1, ..., 9. The decay rate λ is usually set to 0.3 [127]. Fig. 7.1 shows its decay effect.

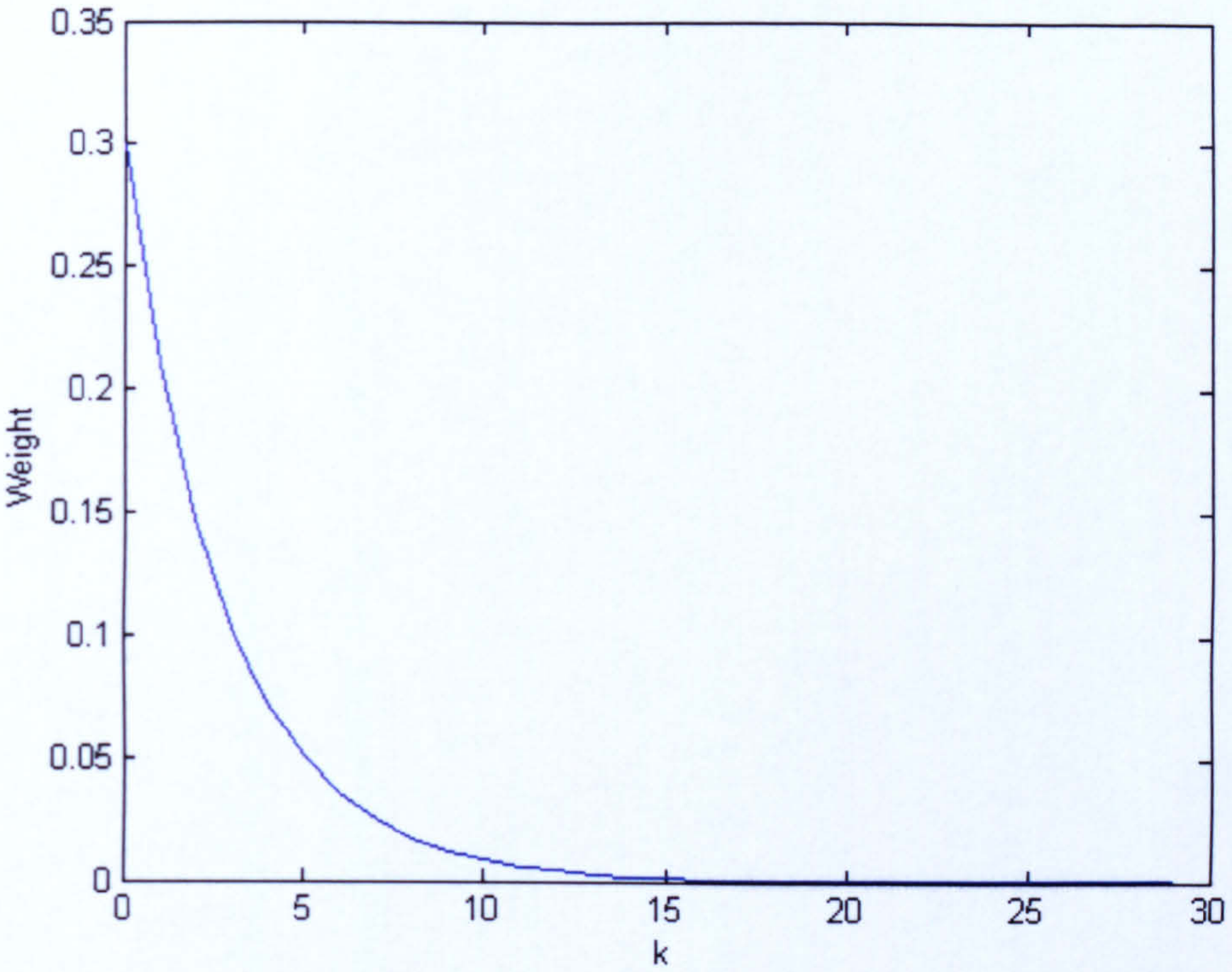

Fig. 7.1 The decay effect with λ set to 0.3.

For each printing operation, we obtain a vector of $\{X_0, X_1, ..., X_9\}$ based on equation (7-2). Given the following stream of printing events, we get the observation value as recorded in Table 7.1:

j = 0, 1, 2, 3, ...

Print 34 pages, Print 172 pages, Print 8 pages, ...

Table 7.1 Observation values for vectors of $\{X_0, X_1, ..., X_9\}$.

j	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
0	0	0	0	0	0	0	0	0	0	0
1	0	0.3	0	0	0	0	0	0	0	0
2	0	0.21	0	0	0	0	0	0	0.3	0
3	0.3	0.147	0	0	0	0	0	0	0.21	0
...	...	...	...	...	...	...	...	...	...	...

Note: At j = 1, '34 pages' falls into group 1 (21-40 pages); at j = 2, '172 pages' falls into group 8 (161-180 pages); at j = 3, '8 pages' falls into group 0 (1-20 pages).

Mike's long-term profile of normal activities is represented by the estimated vector of $\{\overline{X_0}, \overline{X_1}, ..., \overline{X_9}\}$. It is obtained from the training data by averaging all observed vectors of $\{X_0, X_1, ..., X_9\}$. Considering that events in a distributed system actually do not arrive at once but sequentially, we use the following recursive formula to incrementally update $\overline{X_i}$ after each observation [159]:

$$\overline{X_{j,i}} = \frac{(j-1)\overline{X_{j-1,i}} + X_{j,i}}{j} \tag{7-3}$$

where j is the index of the current observation. Eventually, the format of Mike's profile is like:

{Printer, print, (0, 0.055, 0.301), (1, 0.144, 0.254), (2, 0.104, 0.038), (3, 0.054, 0.001), (4, 0.164, 0.025), (5, 0.076, 0.001), (6, 0.111, 0.094), (7, 0.099, 0.266), (8, 0.110, 0.019), (9, 0.056, 0.001), 3.231758}

As we can see, each group is composed of three variables, e.g. (0, 0.055, 0.301). They represent the group number, expected value and current observed value, respectively. The last item in his profile, 3.231758, indicates the threshold value for his printing operation. We will explain it in the following part. For each printing event in the testing data and the corresponding observed vector of $\{X_0, X_1, ..., X_9\}$, we compute X^2 (i.e. D in equation (7-1)) as follows:

$$X^2 = \sum_{i=0}^{9} \frac{(X_i - \overline{X_i})^2}{\overline{X_i}} \tag{7-4}$$

The computed X^2 is small if the observed vector is close to the expected vector.

Similar to the paper [158], in our study we use $\overline{X^2} + 3S_{X^2}$ as the threshold value. We use the training data to estimate the average ($\overline{X^2}$) and the standard deviation (S_{X^2}) respectively, and then load the testing data into the system. If for an event record the calculated value of X^2 is higher than the threshold, we signal this event as an anomaly. Let N denotes the number of records in the training data. The standard deviation S_{X^2} is calculated by:

$$S_{X^2} = \sqrt{\frac{1}{N-1}\sum_{m=1}^{N}(X_m{}^2 - \overline{X^2})^2} \tag{7-5}$$

It is possible that some audit events do not appear in the training data but occur in the testing data. For example, Mike may print 300 pages in his future

operations. Hence, the expected value for such an event is zero after the training. To avoid having a zero at the denominator of equation (7-4), we use the recorded smallest value of 0.001 to replace zero.

Apart from the printed page number, other parameters such as the amount of data transferred and processing time occupied by each session are also monitored and taken into account in a similar way.

7.2 Experiments and results

Again, we use the same simulation environment built with the Georgia Tech Network Simulator (GTNetS) [58], as defined in section 5.2.1. Fig. 5.6 shows the snapshot of the simulated environment.

The first experiment is to examine the false positive rate of SUIDS. We divide the collected audit data into two parts. The first half is used for training, i.e. creating a user's profile. The second half is used for testing. During the test, once the calculated value of X^2 exceeds the threshold, an alarm will be raised. Because the audit data is generated and collected under normal circumstances, any alarm raised during this experiment is considered as a false alarm. The false positive rate is calculated by:

$$R_{fp} = \frac{N_a}{N_e} \tag{7-6}$$

where R_{fp} is the false positive rate, N_a is the number of false alarms that have been raised, and N_e is the number of events that are generated by the legitimate sessions and checked by SUIDS.

There are 3047 records in the training data and 3028 records in the testing data. After the training, we got the value of the average ($\overline{X^2}$) and standard deviation (S_{X^2}) for each type of event. Fig. 7.2 shows the calculated X^2 for the 3028 testing records. We can see that 10 of them have extraordinarily higher values than the others. They are the events that only appear in the testing data. In the end there are total 99 false alarms raised during the test. So the false positive rate of SUIDS is 3.27% (= 99/3028*100%).

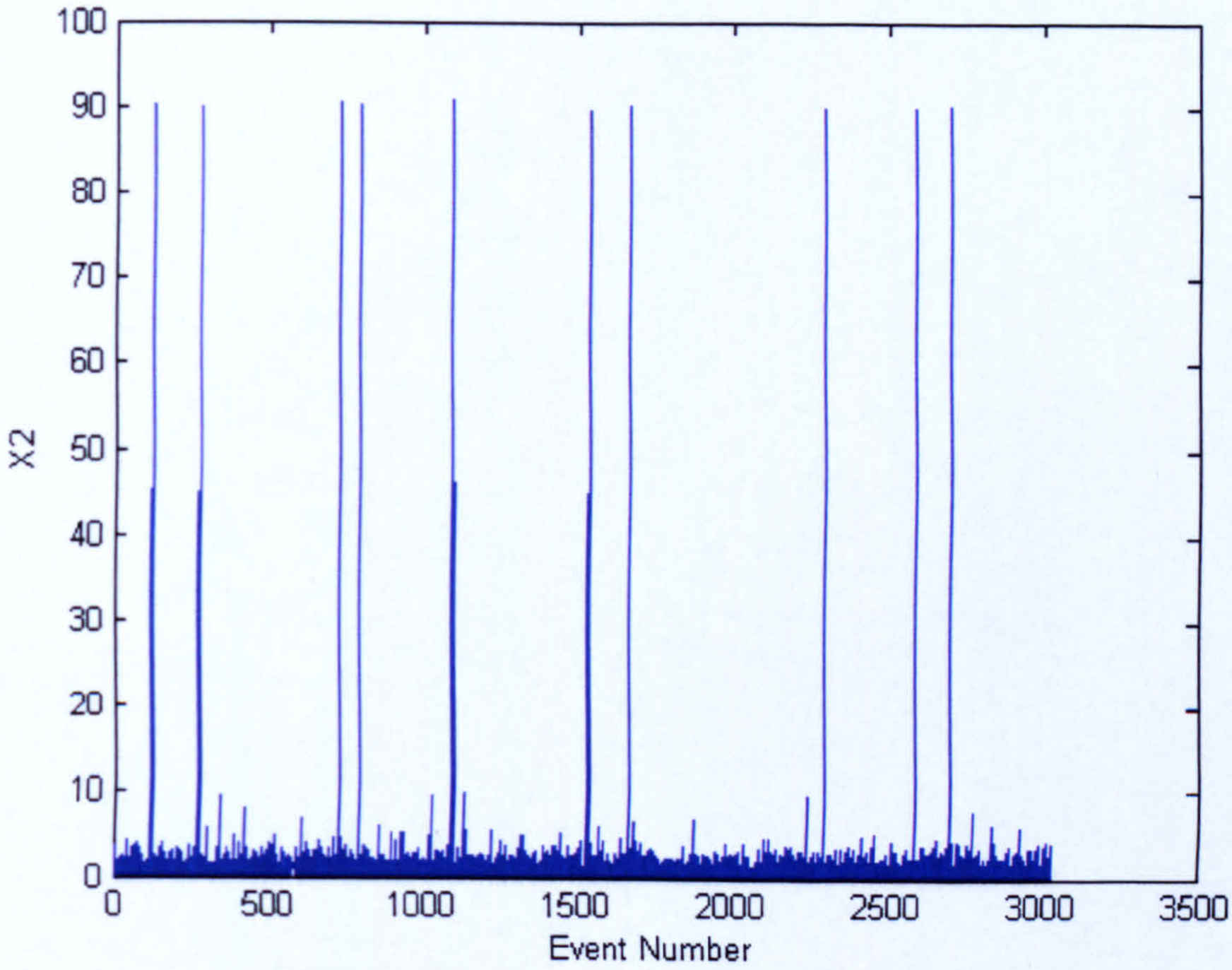

Fig. 7.2 Values of X^2 for normal data.

The second experiment is to test the system's effectiveness on detecting anomalies. The effectiveness of SUIDS is represented by a hit rate. If an alarm is raised in connection with an event record, this record is regarded as being 'hit'. A high hit rate on anomalous event records is preferred. The equation to calculate the hit rate is:

$$R_h = \frac{N_a'}{N_e'} \tag{7-7}$$

where R_h represents the hit rate, N_a' denotes the number of genuine alarms that have been raised, and N_e' denotes the number of events that are generated by malicious sessions and examined by SUIDS.

It is worth noticing that the definitions of N_e and N_e' in equations (7-6) and (7-7) are slightly different from other IDSs. Normally people distinguish event records into normal and abnormal ones based on their characteristics, but we found that on some occasions it is a controversial issue to assert if a record is anomalous. For example, a malicious user always needs to do some preparation work before launching an attack. Although these preparations are closely related

to the attack, its event records could appear normal, as long as they are not against any detection rule. To clarify this issue, we categorize event records based on the nature of the sessions they belong to. A session starts with a service request from a user node, and ends when the service thread is terminated. In this case, all records from a legitimate session will be classified into those counted for N_e. Actually, because in reality we do not know whether a record is anomalous in advance, this generalized classification could be more practical for post-analysis such as tracing the behaviour of an attacker.

We collected another set of data as anomalous data. Two types of anomalies or attacks are introduced into this data set. The first type is a denial-of-service attack, which is generated by deliberately occupying the CPU time of a service node. The second type is a SYN flood attack [143]. In the SYN flood attack the attacker sends TCP/IP SYN (synchronize/initialization) packets, with erroneous return IP address information, to the target. Each SYN packet is a request to open a TCP connection. The victim responds with a SYN/ACK (synchronize/ acknowledge) packet and waits for a response. Soon it will get slowed down as more traffic floods in. In both cases the attacker randomly picks up a service node from the simulated environment as a victim. There are total 2596 records in the anomalous data set.

Fig. 7.3 shows the values of X^2 for the entire anomalous data set, and Fig. 7.4 picks part of the results in order to give a clear view. There are 1199 alarms raised during this experiment. The hit rate is quite low, just 46.19% (= 1199/2596*100%). As we explained earlier, it is caused by the fact that not all the records in a malicious session act against the rules being checked by SUIDS. A specific attack will only show anomalies in certain aspects, e.g. an anomalous traffic pattern or processing time. Actually, if we measure the system's hit rate by excluding the accessory records, we will get a so-called 'key anomaly' hit rate. The key anomalies are identified according to each attack's main influence on the system being protected. For example, in our experiments the DoS attack directly affects the CPU processing time and the SYN flood attack introduces unusual traffic patterns. We addressed 982 key anomalous records from the anomalous data set, and 924 of them triggered an alarm. So the key anomaly hit rate of SUIDS is 94.09% (= 924/982*100%).

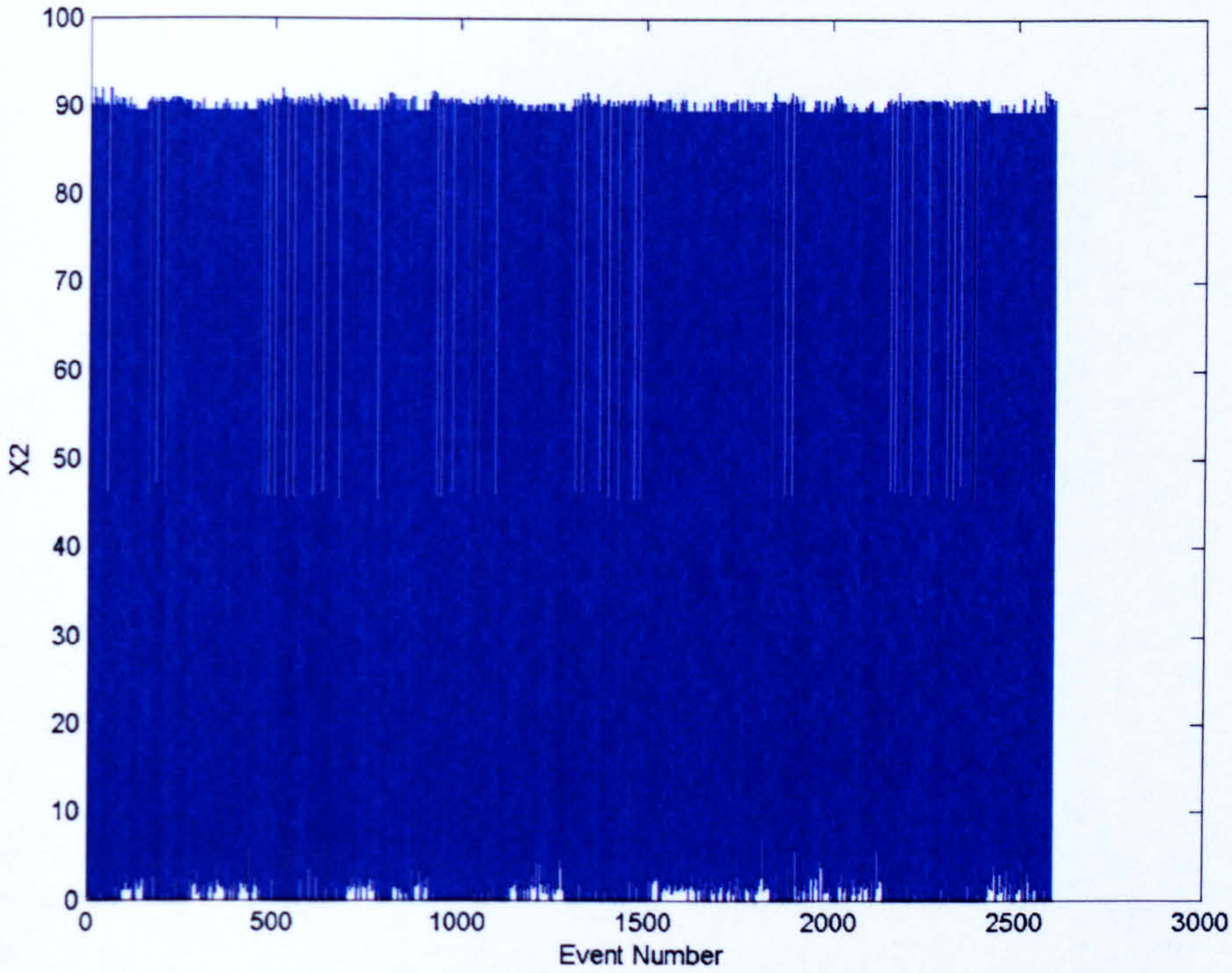

Fig. 7.3 Values of X^2 for anomalous data with all the results for events 1-2596.

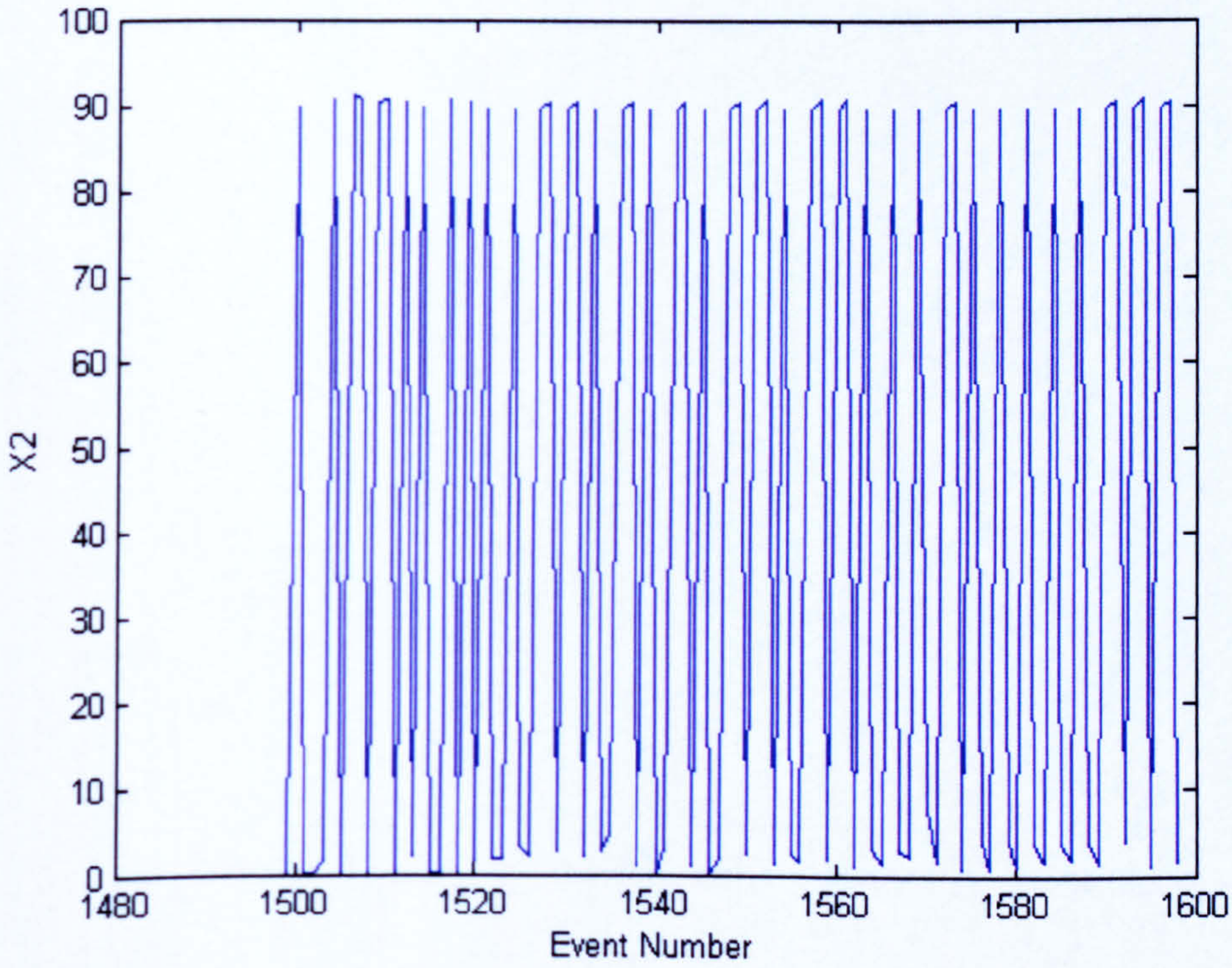

Fig. 7.4 Values of X^2 for anomalous data with partial results for events 1501-1600.

To further prove the effectiveness of SUIDS, we also measured the system's hit rate by sessions other than records. Similar to the definition of the hit rate by records, if there is an alarm triggered in connection with an intrusion session, we signal this session as being 'hit'. The result is calculated by dividing the number of signalled malicious sessions by the total number of them. As long as no malicious session has been ignored by SUIDS, we can regard the system as being secured. This measurement will not affect the real time feature of SUIDS as the detection mechanism still works based on the stream of event records. 2596 anomalous records are generated by 789 intrusion sessions. After the experiment, all these 789 sessions have been signalled. The system's hit rate by sessions achieved 100% (= 789/789*100%).

Instead of using the pure normal or pure anomalous data set, in the last experiment we tested SUIDS with a mixed data set by combining normal activities and anomalies. The audit data we used here are the same as those used in experiments 1 and 2. We chop the anomalous data into several portions and insert them into normal data stochastically. The false alarm rates and hit rates (by both records and sessions) are calculated in the same way. Fig. 7.5 shows the calculated values of X^2 for the mixed data and Table 7.2 shows the experimental results. We can see from Table 7.2 that the false alarm rate here is little higher than in experiment 1. It indicates that introducing anomalies into normal data do affect the final result of R_{fp}. However, the malicious session hit rate still keeps 100%.

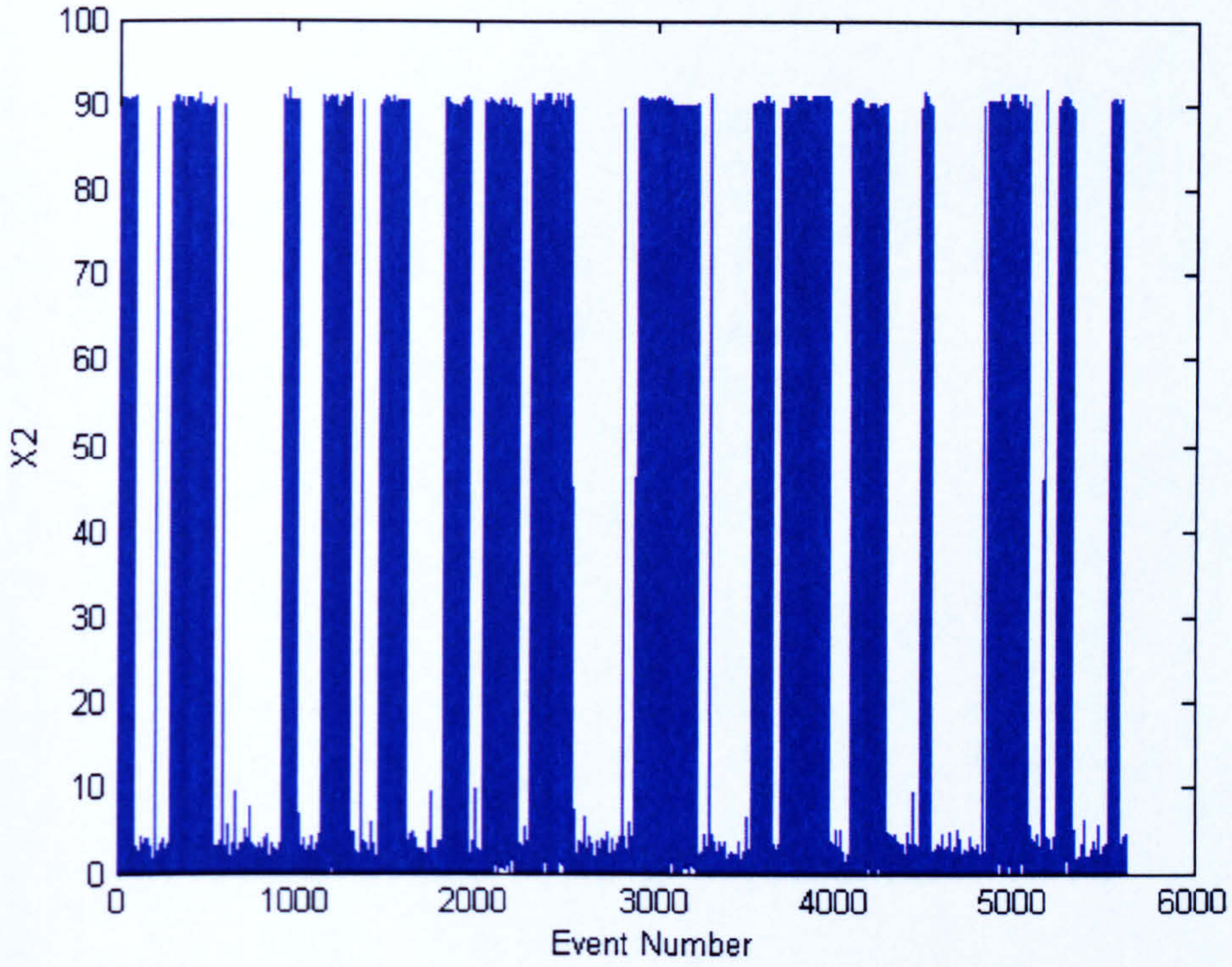

Fig. 7.5 Values of X^2 for mixed data.

Table 7.2 False alarm rates and hit rates (by both records and sessions) for a combined data set.

False Alarm Number (N_a)	120
Normal Record Number (N_e)	3028
False Alarm Rate (R_{fp})	3.96%
Genuine Alarm Number (N_a')	1171
Anomalous Record Number (N_e')	2596
Hit Rate by Records (R_h)	45.11%
Signalled Key Anomalous Record Number	919
Key Anomalous Record Number	982
Key Anomaly Hit Rate	93.58%
Signalled Malicious Session Number	789
Malicious Session Number	789
Hit Rate by Session	100%

In most cases, we want to keep a high hit rate on anomalous records/sessions since any ignored attack may cause severe damages to the entire system. The system's tolerance on the false alarm rate depends on individual requirements. Because the false alarm rate of SUIDS is very low (< 3.96%), we can conclude that SUIDS can achieve a good performance by applying this altered chi-square statistic test.

Comparing with the string-based approach proposed in chapter six, the detection effectiveness of the chi-square statistic test shows little improvement. The chi-square statistic test has a better false alarm rate, while the string-based approach has a slightly better hit rate. Both of them are lightweight and able to detect anomalies in real-time. However, because the user profile used for the string-based approach is bigger than that for the chi-square statistic test, the former requires more system resources. Furthermore, since the string-based approach is based on a cumulated result, it needs an extra process to identify a malicious event record. Thus we think the chi-square statistic test is a better solution in general.

7.3 Summary

In this chapter, we presented an improved detection method for SUIDS. It adopts a chi-square statistic test to calculate the deviations between a user's short-term behaviour and his long-term profile. The experiment results encouragingly show that SUIDS can achieve a high hit rate on anomalous records/sessions with a maximum false alarm rate of only 3.96%. Based on this effective detection method, in the next chapter, we will investigate the possibilities to make SUIDS more resource-efficient. A resource-efficient detection scheme will help reducing the usage of CPU and storage space. More importantly, for ubiquitous networks which may contain many battery powered devices, reduced energy consumptions will extend the lifetime of the entire network.

CHAPTER EIGHT

ACHIEVING ENERGY-EFFICIENCY IN SUIDS

In the last two chapters we examined two different detection methods for SUIDS. The inherent features of ubiquitous computing request SUIDS to give special concern to the issue of resource-efficiency. In this chapter, we present a comprehensive analysis of energy consumed in SUIDS and propose a profile splitting technique in order to reduce the energy consumption. Specifically, it shows how a head node can be utilized to save the computing-related energy; how a user profile can be managed in a distributed pattern to reduce the communication-related cost; and how a hybrid metric is used to balance both of them in order to extend the network lifetime.

8.1 Energy-efficiency in SUIDS

System resources are crucial in ubiquitous networks. Ideally, during its implementation, SUIDS needs to balance many factors such as CPU processing speed, storage space, trustworthiness and etc. In the past decades, the CPU processing ability and storage space of computer systems keep fast growing, by obeying Moore's Law [106]. Battery capacity becomes a bottleneck for most battery-powered devices. We think in the foreseeable future, the energy issue will remain as a crucial hurdle on the road towards ubiquitous computing. Hence, in this chapter we particularly focus on saving energy. We analyze the energy consumed by SUIDS and present a new approach to reduce it. Other factors will be considered in the next chapter.

8.1.1 Energy consumptions in SUIDS

The energy consumed by SUIDS can be classified into two categories: communication-related energy and computing-related energy.

Communication-related energy refers to the energy used by the radio transceiver of a node to communicate with others. The contents of the communication include transmitted/received event records and user profiles. An approximation of energy consumption when transmitting or receiving r bits between two nodes n_1 and n_2 with a distance of $d(n_1, n_2)$ is given in [111] as:

$$E_{tx} = (\alpha_{11} + \alpha_2 d(n_1, n_2)^n) r \quad (8\text{-}1)$$

$$E_{rx} = \alpha_{12} r \quad (8\text{-}2)$$

where E_{tx} denotes the transmitting energy and E_{rx} denotes the receiving energy. α_{11}, α_2 and α_{12} are constants, and their typical values are $\alpha_{11} = 45nJ/bit$, $\alpha_{12} = 135nJ/bit$, $\alpha_2 = 10pJ/bit/m^2$ (for $n = 2$) and $0.001pJ/bit/m^4$ (for $n = 4$). n is the attenuation factor and in this study we use $n = 2$.

The computing-related energy refers to the energy used to implement the intrusion detection modules. It is mainly dedicated to monitor network status and user activities, execute intrusion detection algorithms, maintain and update user profiles. The calculation of the computing-related energy is a very complex task. A more detailed definition and simulation model are needed. In this thesis, we assume this part of the consumption is proportional to the number of event records. That means the more users' activities are observed, the more computing-related energy will be consumed. For each record, processing it is a fixed charge ($5mJ$), regardless where detection modules are.

8.1.2 Save computing energy by using head nodes

As we explained in chapter five, the network nodes in our system are categorized into head nodes, service nodes and user nodes. Head nodes, for example PCs, normally have no constraints on energy when they serve as fixed workstations. In our original design, a user profile in SUIDS follows the user around and stays at the nearest head node to the user. Head nodes are in charge of receiving event records from service nodes and detecting anomalies. In this way, most of the computing-related energy is consumed at head nodes and can be omitted since the head nodes have unlimited power supplies.

However, just like today's ad-hoc networks, the great feature of 'anytime and anywhere' inevitably constrains the availability of head nodes in ubiquitous networks. Sometimes we may not be able to find a suitable head node to host a user profile. And even if a head node is available, more energy could still be consumed on communications if it is too far away from service nodes. In order to extend the lifetime of a ubiquitous computing network and cope with the situation where no head node is available, in the next subsection we will present a distributed profile splitting technique to replace the centralized model defined in chapter five.

8.1.3 Save communication energy by splitting user profiles

To save the energy and time spent on communications, service nodes need to participate in intrusion detection in a more proactive way. In this chapter we try to achieve energy efficiency by arranging the detection modules and user profiles of SUIDS in a distributed pattern. Obviously, if event records are processed locally instead of sending them to head nodes, energy consumed on data transmission will be reduced.

A user's profile in SUIDS is constructed by a list of entries. Each entry records a user's behaviour regarding his usage of a particular service. The structure of an entry is like:

{Service-ID, Action-Type, (Parameter Sample Spaces, Estimated Value, Observed Value), Threshold}

We are inspired to split the user profile into smaller parts based on the Service-ID and distribute them to the corresponding service nodes. Because the entries in a user profile are independent of each other (according to the detection method of SUIDS), splitting the user profile will not affect the result of intrusion detection. When a user requests a service, the related service node will get the corresponding entries from the user profile. The service node will calculate the value of X^2 (the measurement of similarities between the expected and observed values linked to the node) locally and send the updated entries back to the head node when the user moves to other domains. In this way, only a small part of the user profile needs to be transmitted between the service node and its head node.

8.1.4 Choose proxy nodes based on a hybrid metric

Processing event records locally means that the detection algorithm of SUIDS will be executed at service nodes. The prerequisite for this method is that the service nodes are able to afford this extra load. Unfortunately, in ubiquitous networks it is not always the case. A service node might be constrained by its limited battery capacity. The overuse of the service node may cause battery exhaustion and shorten the service lifetime. Hence, another process is needed to choose the most suitable place to allocate the split user profile and execute the intrusion detection algorithm. We will use the term 'proxy node' to denote a dedicated place/node for this purpose.

Possible choices of proxy nodes for a service node include the service node itself and other nodes around it within one hop. Although no solution has been proposed yet to address the same issue in the area of intrusion detection, we realized that, to some extent, our work can benefit from existing research in the area of energy-efficient routing in mobile ad-hoc networks [95, 136, 151]. In this chapter we use the following metrics to choose a proxy node:

1. Minimum transmission power E_{tx}. This metric tries to find the most efficient proxy node in terms of saving communication energy. Because for the same amount of data, the energy consumed on receiving (E_{rx}) is unchanged, we only have to compare different amounts of transmission energy (E_{tx}) related to possible proxy nodes. Equation (8-1) shows that E_{tx} consumed between a service node and a proxy node depends on d (d is the distance between the service node and proxy node). As a feature of ubiquitous computing, the physical positions of service and proxy nodes can be used as available information. Obviously, for a service node, this metric will always lead to the proxy node that is closest to but different from the service node (with the minimum d).
2. Maximum residual energy B_t. Although the minimum transmission power E_{tx} may reduce the total energy consumption, it does not reflect directly on the lifetime of each node. If a service node chooses a node with less residual energy, the selected node will die of battery exhaustion sooner. Therefore, the remaining battery capacity of each node is a more accurate metric to describe their lifetimes. This metric prefers the proxy node with

the maximum residual energy at time t (B_t). The target of this metric is to evenly distribute energy consumptions among network nodes and extend the network lifetime.

3. Minimum energy consumption rate $(B_0-B_t)/t$. Residual energy B_t represents a node's current condition at time t. It cannot reflect the node's past and future usage trend. Because the intrusion detection module of SUIDS introduces extra burdens on both communication and computing, an energy consumption rate is also an important metric to be considered. Let B_0 denote the initial battery capacity of a proxy node, and B_t denote its residual energy at time t. Assuming the energy consumption rate is a constant value, $(B_0-B_t)/t$ represents how busy/active this node is in the network. This metric can be regarded as a complement to the second metric.

The above three metrics are not consistent with each other. For example, in Fig. 8.1 service node 1 will work out different proxy nodes when applying these three metrics respectively. Metric 1 will choose node 2 as the proxy node of node 1 since it consumes the least transmission power. Metric 2 will choose node 3 because it has the most residual energy. And metric 3 will choose node 4 since it consumes energy at the lowest rate. It is worth to notice that metrics 2 and 3 do not necessarily mean a longer network lifetime. In some cases it might even get the opposite result if they pick up a proxy node that consumes too much transmission power.

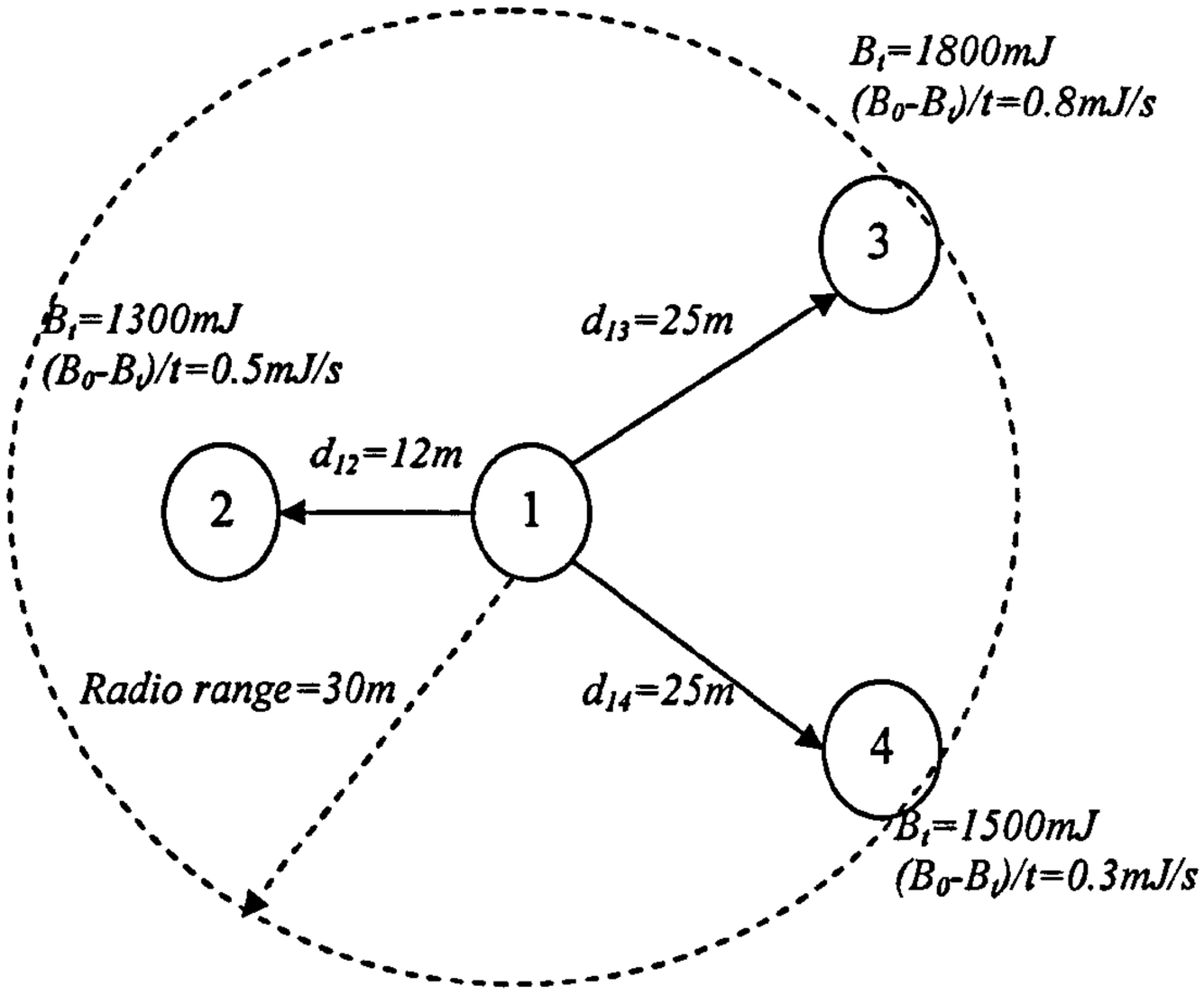

Fig. 8.1 Three metrics will work out different proxy nodes.

In order to balance these three metrics, in this chapter we use a new conditional hybrid metric. Basically, from all the candidate nodes, the one with the maximum value of

$$M_h = \frac{B_t - \beta_1}{(\frac{B_0 - B_t}{t} + \beta_2) \times E_{tx}} \tag{8-3}$$

will be chosen as the proxy node. M_h is referred to as the conditional hybrid metric. Parameters β_1 and β_2 are the conditions. β_1 works as a threshold to rule out a set of nodes with less residual energies. β_2 sets a minimum expectation of the energy consumption rate. By adjusting the values of β_1 and β_2, we will be able to prevent some extreme cases. For example, if a node keeps idle for a long time and has very little energy left, without β_1 and β_2 (set to 0), it might be undesirably selected as a proxy node. In this study we set $\beta_1 = 20mJ$ and $\beta_2 = 0.05mJ/s$.

8.2 Experiments and performance analysis

8.2.1 Modified simulation environment

Because GTNetS does not provide the functionality for measuring energy consumption, we chose another simulator - Georgia Tech Sensor Network

Simulator (GTSNetS) [111] - to prove the effectiveness of our method. GTSNetS is a fully-featured sensor network simulation tool based on the GTNetS. The main difference is that it provides each sensor node with a simulated battery in order to measure its energy consumption. Because GTSNetS was dedicated to sensor networks, we have to modify the source of the simulator in order to make it fit for our experiments. The major modifications we made include:

- Inherit and transform basic sensor network applications to CBR and TCP applications. The CBR application is used to generate constant bit rate data between two nodes. The TCP application creates a simple model of a request/response based TCP session. A TCP server is bound to a specified port, and listens for connection requests from TCP peers. The data received from the peers specifies how much data to send or receive.
- Disable the sensing function of sensor nodes and transform the simulated environment of a sensor network into that of a simple wireless ad hoc network. Along with the alterations to applications and network nodes, the routing protocol used has also been changed from Directed Diffusion [73] to DSR [81].
- Generate our own trace files by using a timer variable to check node states every 1.5 seconds. If a node's residual energy is less than $1mJ$, the node will be considered as dead due to its battery exhaustion. To simplify the issue, we assume that if the number of dead nodes exceeds half of the total node number, the network will be considered as dead too.

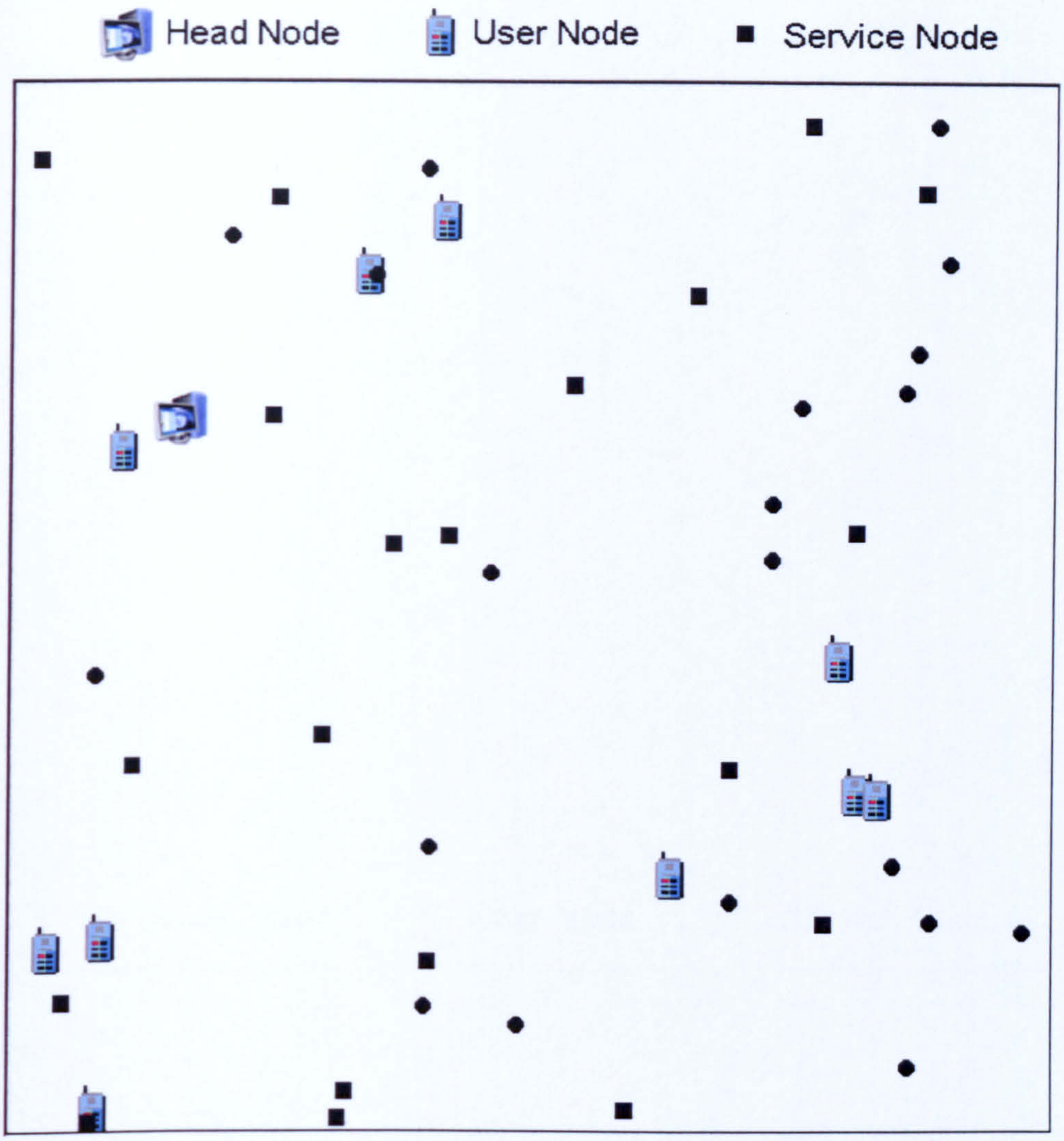

Fig. 8.2 Modified simulation environment with GTSNetS.

The simulated network has total 51 nodes in a 120x120m^2 area. Initially, there is one head node, ten user nodes and forty service nodes. All the nodes in our simulations are connected and communicate with each other through wireless connections, i.e. in an Ad Hoc pattern. The default routing protocol is DSR. Fig. 8.2 shows a snapshot of the simulated environment. We assume the signal transmission medium is homogeneous, i.e. fixed α_{11}, α_2 and α_{12} with $n = 2$ in equations (8-1) and (8-2), and all the nodes have the same radio range (30m).

There are some factors that will affect the performance of SUIDS in terms of energy-efficiency. In the next few subsections, we analyze each of them and demonstrate their influences on the network lifetime.

8.2.2 Effect of the hybrid metric

The first experiment is to examine the effect of the hybrid metric defined earlier. We created three scenarios. In the first scenario, all event records will be processed locally at the service nodes. In this way, the communication-related energy is much reduced. In the second scenario, the service nodes will always choose the head node as a proxy node. All the event records will be analyzed at the head node. Thus the computing-related energy can be reduced. In the last scenario, once a service node has been activated, all nodes within its radio range, including itself, will be examined against the hybrid metric. The node with the highest value of the metric will be chosen as its proxy node. The event records of the service node will be sent to the proxy node and processed there. The TTL (time to live) field of the request message sent by service node set to 1 (hop) in order to reduce the amount of communications. Only a small amount of data (the value of hybrid metric) needs to be sent back to the service node during the proxy selection phase.

In all these three scenarios, the communication-related energy is calculated based on equations (8-1) and (8-2). The computing-related energy is shared by the service and proxy nodes. We assume the energy consumed at a service node is proportional to the length of an event record it generated, and that at its proxy node is a fixed cost for the record reception and processing ($5mJ$). The simulation will end after the network has died (i.e., over half of the service nodes are battery exhausted). The purpose for this is to examine how the introduction of the hybrid metric will affect the network lifetime.

We tested each scenario with different user nodes and took mean values as the final results. Fig. 8.3 shows their differences. The horizontal axis denotes the number of dead nodes and the vertical axis denotes their death time. At the node number equal to 20, the network is dead. Not surprisingly, with our hybrid metric, the system has the best performance compared with simply using a service node or head node as a proxy for event record processing. The impact of the hybrid metric gradually improves as the simulation proceeds. The average node lifetime is increased from 4488.525s (scenario one) and 4563.89s (scenario two) to 5845.595s (increased 30.23% and 28.08%, respectively). The network lifetime is

extended on average from 6314.6s (scenario one) and 7042s (scenario two) to 8736.1s (increased 38.35% and 24.06%, respectively).

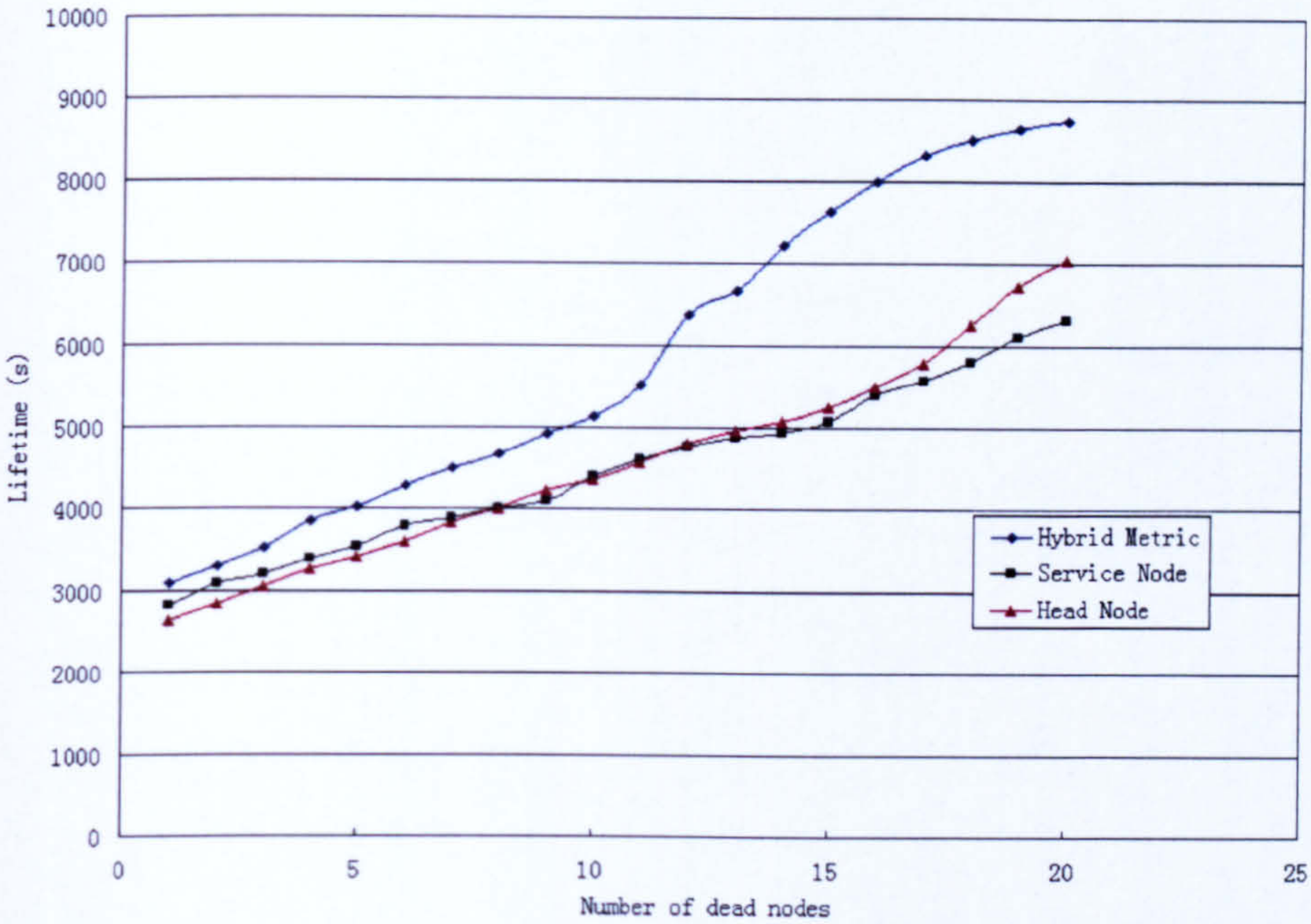

Fig. 8.3 Impact of the proposed hybrid metric.

8.2.3 Head nodes' density and distribution

In the first experiment, we assume that there is only one head node available in the simulated environment. Certainly, if more head nodes are deployed in the network, the result could be different. In the second experiment we examine how the existence and deployment of head nodes will affect our system.

We increased the number of head nodes from one to five and reran the simulations, respectively. To keep the total node number unchanged, we reduced the number of user nodes correspondingly. In the case of five head nodes, only six user nodes are left. In order to ensure a fair comparison, all the results are obtained by using the same set of user nodes. The head nodes are randomly located and one of them will be chosen as a proxy node. Because they all have unlimited power supplies, the hybrid metric cannot be used here. In this experiment we use the distance from a service node to a head node as a metric to measure its suitability as a proxy node. Normally, the closer a head node to an activated service node, the less energy consumed on the communication between

them. Hence, a service node will always choose the closest head node as its proxy node. Experiment results are shown in Fig. 8.4. Basically, since the additional head nodes are deployed, the network lifetime is generally extended. The averaged node lifetimes are 4368.958s, 4618.675s, 4782.65s, 4744.051s, and 5387.984s for the number of head nodes from one to five, respectively. However, even with the five head nodes (almost 10% of the total node number), the performance of using the head nodes is still no better than using our hybrid metric which has the averaged node lifetime of 5449.575s (re-calculated by using the same set of user nodes as well).

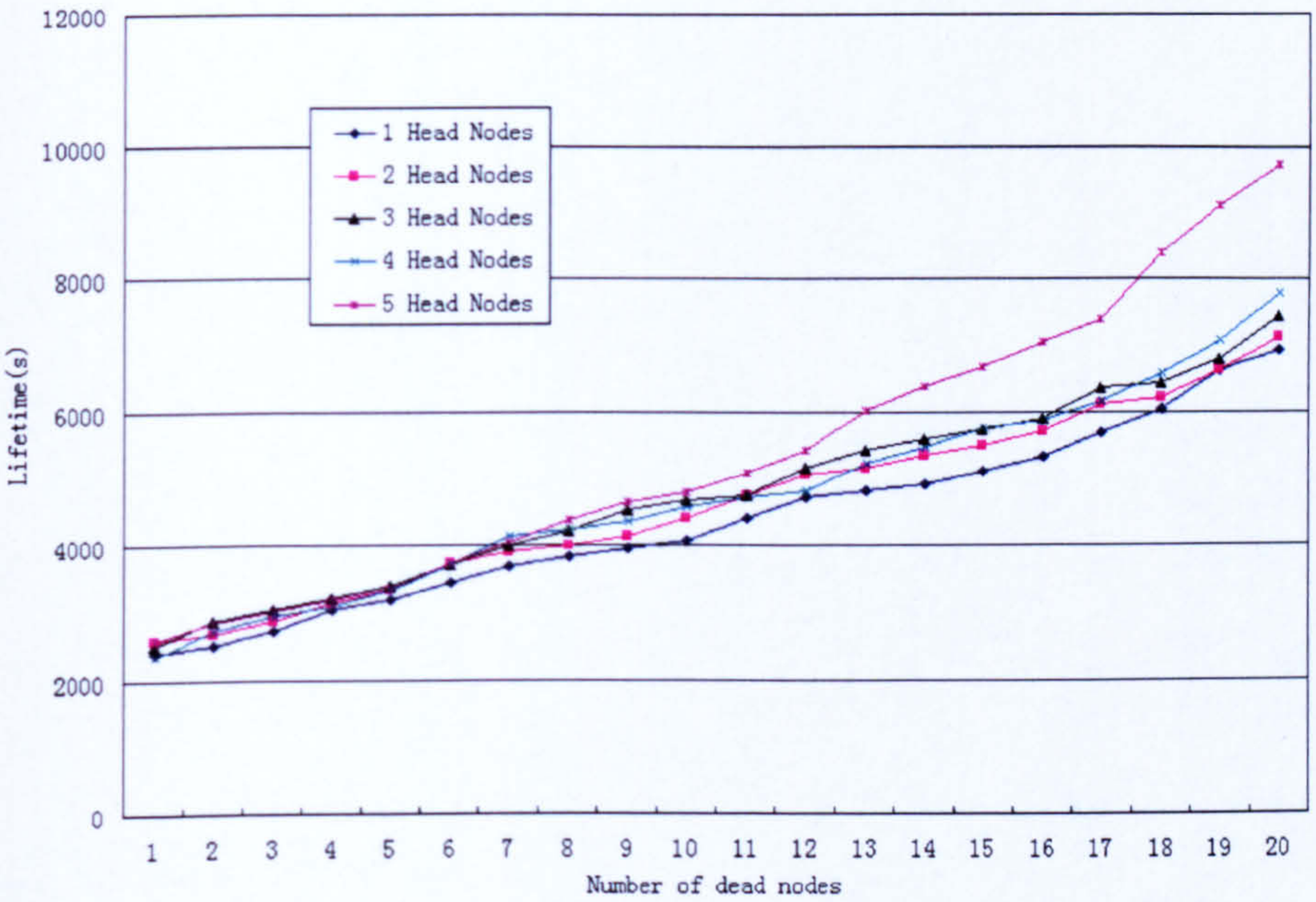

Fig. 8.4 Use of head nodes from one to five.

Apart from the number of head nodes, their locations also play an important role. If a head node is easy to reach from a service node, the energy consumed on their communication could be reduced. In the last test, we choose the case of four head nodes and deploy them uniformly instead of stochastically. The network is equally divided into four squares and the head nodes are deployed at the center of the squares. The experiment results are shown in Fig. 8.5. Although the network lifetime is dramatically increased in the case of uniform distribution, in reality the deployment pattern of head nodes is different from case to case. As we mentioned earlier, because the availability of head nodes is not guaranteed in ubiquitous

networks, we think our hybrid metric still has a greater and more generalized usefulness.

Note that in Fig. 8.5 the random distribution overperforms the hybrid metric around 10 dead nodes. It could happen at the early stage of an experiment if there are relatively more head nodes. However, the case with the application of the hybrid metric still has a longer average node lifetime for a long-term observation.

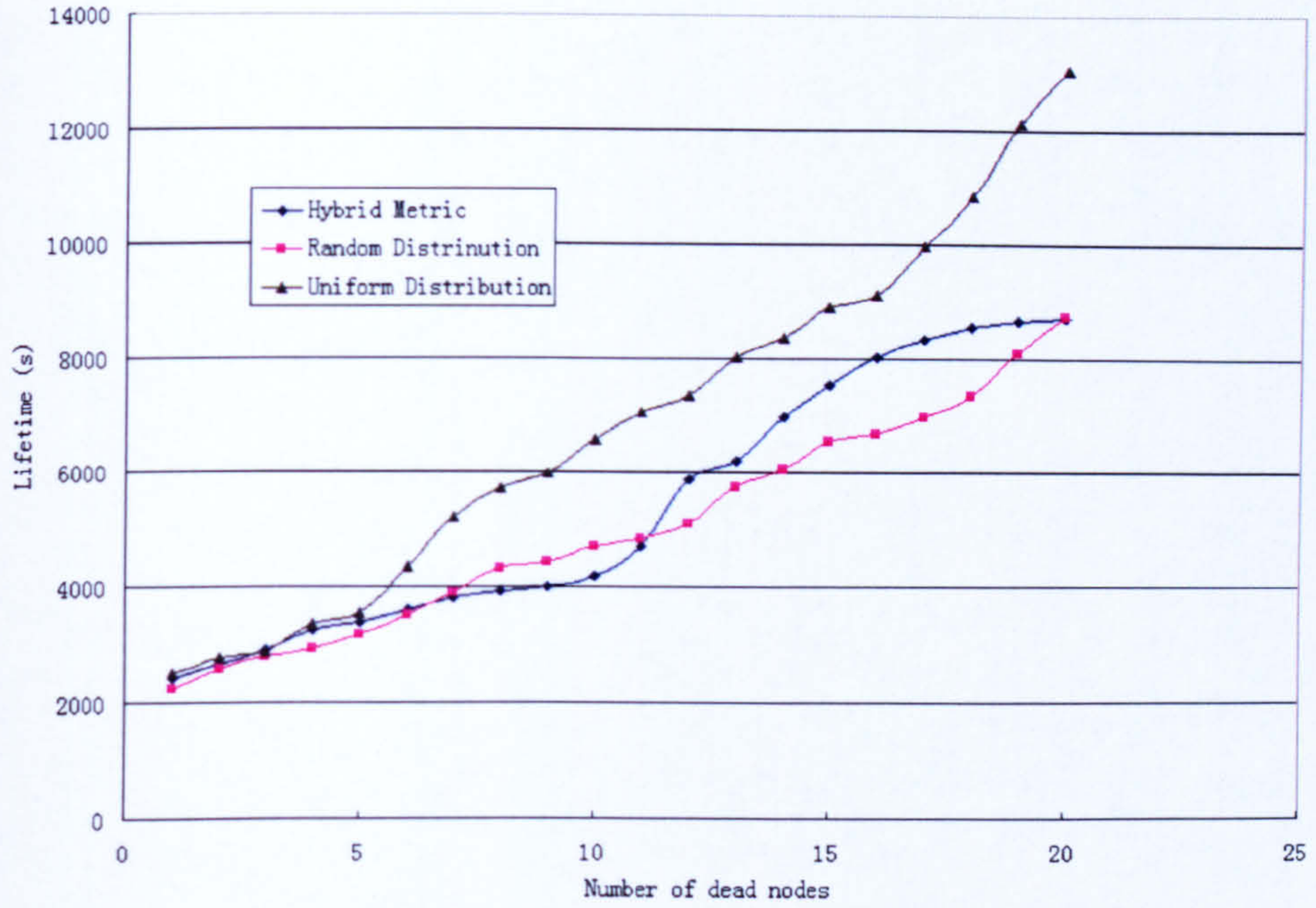

Fig. 8.5 Four head nodes with different distribution patterns.

8.2.4 User nodes' mobility

User nodes in our experiments are mobile. The mobility pattern used here is the Random Waypoint (RWP) model [109]. The RWP model is widely used in the simulations of Ad Hoc networks. There are two factors in the RWP model: a user's velocity and thinking time. Basically, in the RWP model, each node moves along a zigzag line from one waypoint to the next. The waypoints are uniformly distributed over a given area. At the start of each leg a random velocity is drawn from the velocity distribution (in a basic case the velocity is constant 1). The nodes may have so-called 'thinking times' when they reach each waypoint before continuing on the next leg. We cannot control a user's mobility, but the

correlations between the users' mobility patterns and the network lifetime may help us to adjust our strategies.

We first examine the effect of a user's velocity. Let the thinking time be fixed to 600 seconds. Four velocities have been tested: 1m/s (Walk), 5m/s (Bicycle), 10m/s (Motorcycle), and 15m/s (Car). The network lifetimes under the different velocities are shown in Fig. 8.6. We can see that at a low speed (1-5m/s), the network lifetime is shorter than that at a higher speed (10 m/s). It can be explained that with a fixed thinking time, a higher speed scenario covers a wider area during the same period. It tends to give the service nodes more choices on a proxy node and helps the system to distribute its residual energy evenly. However, if the speed continues growing (15 m/s), the extra energy consumed on the dynamic routing will partly leverage the benefit brought by the wider coverage. The network lifetime will be shortened.

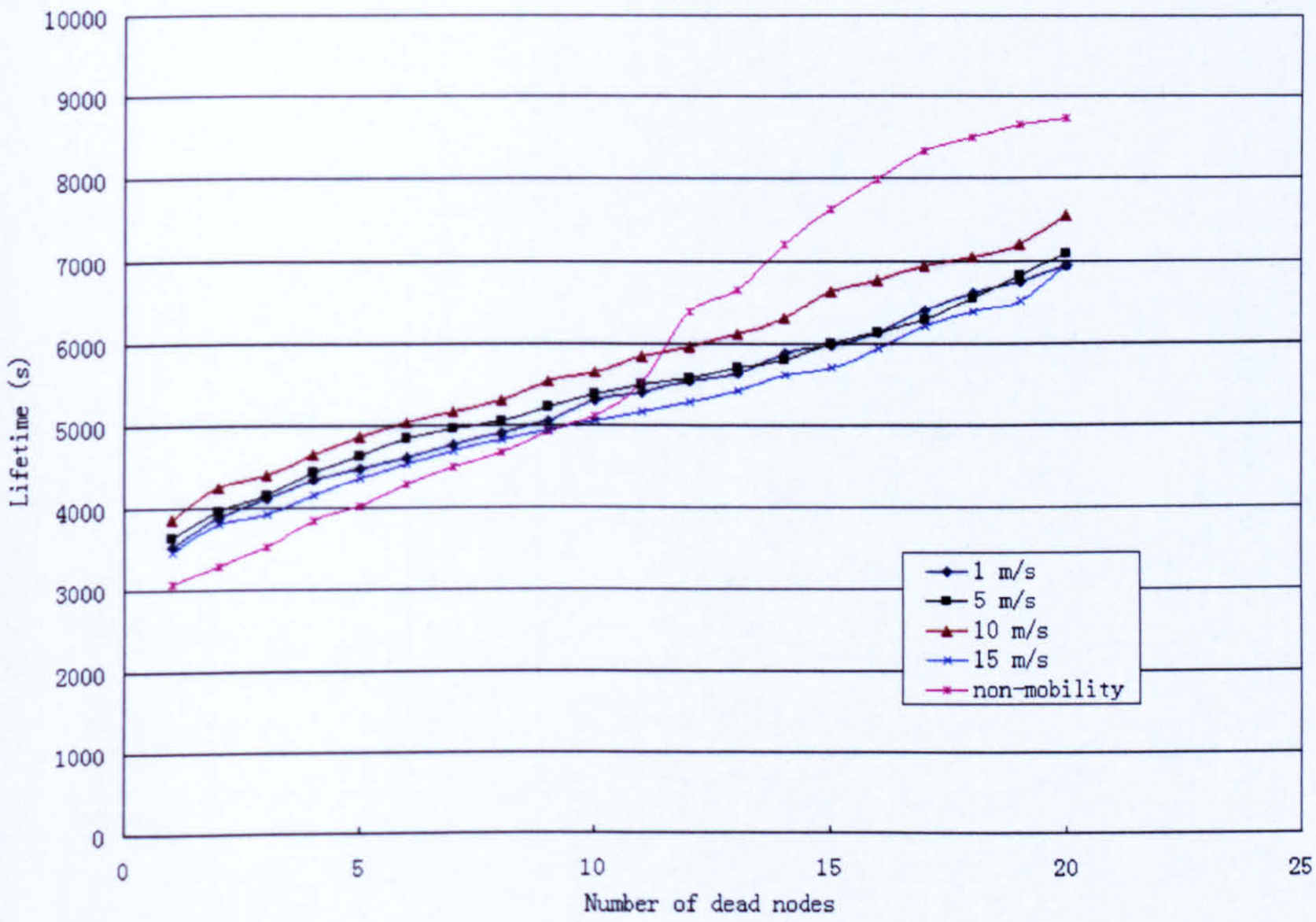

Fig. 8.6 Effects of users' velocities.

Similarly, in the next experiment we tested different thinking times from 300s to 1500s. The velocity is fixed to the most common scenario 1m/s (Walk). The experiment results can be found in Fig. 8.7. Basically, a shorter thinking time has better performance than a longer thinking time. It can be explained that under the

same speed, a shorter thinking time can cover a wider area during the same period.

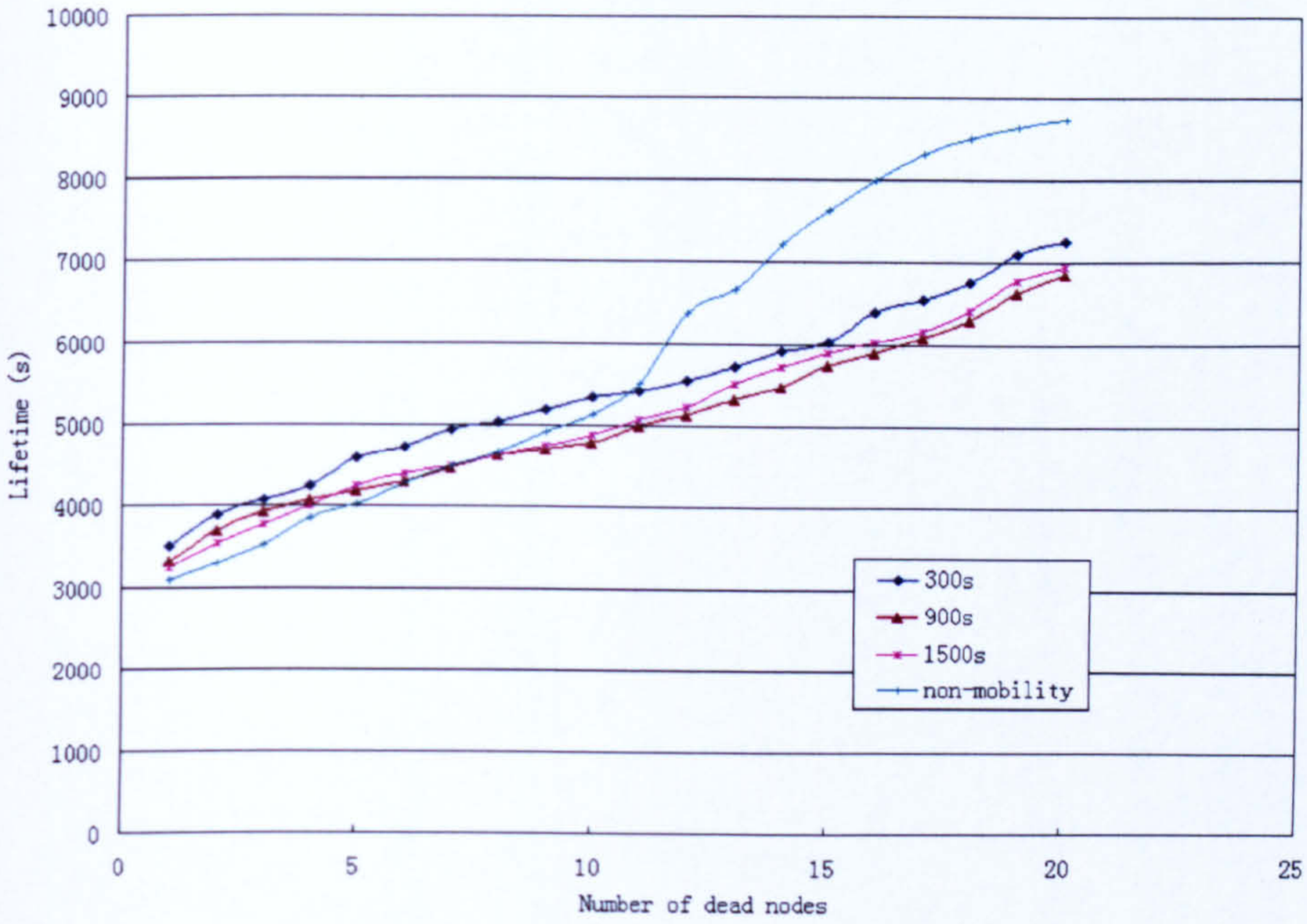

Fig. 8.7 Effect of users' thinking times.

It is worth noticing that in both of figures 8.6 and 8.7, the network lifetime lines are flatter than those in the 'still' case (without mobility). It further proved that though user nodes' mobility may require more energy on routing, it can also help to distribute the energy consumptions evenly among the nodes and extend the network lifetime eventually.

8.3 Summary

In this chapter, we analyzed the energy consumptions in SUIDS for a ubiquitous computing network and categorized them into two parts: computing-related and communication-related. The computing-related part can be reduced by taking advantage of head nodes' unlimited computation supplies; and the communication-related part can be reduced by splitting user profiles and implementing the detection modules of SUIDS locally. To balance these two, we proposed a conditional hybrid metric. By taking various energy-related factors into account, the hybrid metric helps SUIDS achieve better performance in terms

of energy-efficiency. As a result, the network lifetime is beneficially extended. It has to be pointed out that our method is designed for those battery powered devices. Some service nodes such as a smart refrigerator may also have an unlimited energy supply. In this aspect, their capacities are equal to those of head nodes. A combined consideration about the density, distribution and mobility of head nodes may help to deploy the network more effectively in the future. In the next chapter, more factors such as a node's processor speed and available storage space will be considered. Especially, we take the trustworthiness of nodes into account during the selection of a proxy node. In this way, SUIDS is enhanced with stronger security assurance.

CHAPTER NINE

BALANCING INTRUSION DETECTION RESOURCES IN UBIQUITOUS COMPUTING NETWORKS

Resource-efficiency is regarded as a key objective for all applications in ubiquitous computing. In the last chapter, we used an energy-related hybrid metric to reduce the energy consumptions of SUIDS. It balanced the transmission power, residual energy, and energy consumption rate of a node during the selection of a proxy node. Based on this work, in this chapter we present a comprehensive analysis of the resource constraints in SUIDS and propose a new method in order to take other factors such as computing ability, storage space and trust levels into account. Specifically, it shows how a node's computing availability is measured in relation to its energy usage; how a node's trust level is estimated based on multi-factors; and how a hybrid metric is used to balance these concerns together. As a result, SUIDS achieves better performance in terms of resource efficiency together with enhanced security assurance.

9.1 Selecting a proxy node based on additional factors

Overcoming the barriers of limited resources in ubiquitous computing is always one of our main objectives. In this section, we present a new approach to improve the performance of SUIDS regarding its usage on system resources. We use the same profile splitting technique mentioned in chapter eight and an enhanced hybrid metric to select a proxy node. The proxy node is used to perform delegated burdensome intrusion detection tasks. During the selection of the proxy node, there are four key resources to be considered: energy, storage space,

processor speed (busy/idle ratio), and trust. Among them, the energy, storage space and processor speed are quantified metrics. It is possible to compare them directly by knowing their numerical values. In contrast, to measure the trustworthiness of a node, an extra process is needed to evaluate its level first. However, it is important to take the trust into account as the delegation of detection tasks should not introduce new threats or vulnerabilities into the system. In this section, we propose a new approach to measure these resources together. This method exploits the hidden correlations among the resources.

9.1.1 Remaining energy and storage space

A node's remaining energy and storage space are variables. Their values change all the time, depending on the node's current condition as well as the surrounding environment. Sufficient remaining energy and storage space are prerequisites for a node to be chosen as a proxy node. In contrast with consumable energy, occupied storage space can be released once its use is completed. Hence remaining energy is a more crucial factor during the selection of a proxy node.

To simplify the issue, in this study we assume that all the nodes are powered by batteries. It is more like a mobile Ad Hoc network. As we explained in chapter eight, although in reality some devices, such as head nodes, may have unlimited energy supplies, simply relying on them may cause quick battery exhaustion for other nodes. Because remaining energy is just one aspect of the issue, to use the energy smartly and reduce its total consumption, other energy related factors also need to be considered. Here we use the same three factors mentioned in chapter eight: minimum transmission power E_{tx}, maximum residual energy B_t, and minimum energy consumption rate $(B_0-B_t)/t$. The final metric regarding a node's energy is:

$$E = \frac{B_t - \beta_1}{(\frac{B_0 - B_t}{t} + \beta_2) \times E_{tx}} \tag{9-1}$$

9.1.2 Available computing ability

The processor speed of a node/device is a constant. It denotes the node's computing capability. The processor speeds of different nodes vary from several

MHz to several GHz. Normally small devices such as sensors have limited computing abilities as their sizes are confined. However, the availability of a node's actual computing ability is a variable. Its processor speed needs to be considered together with its busy/idle ratio. If the node is always in a busy status, its actual computing ability dedicated to IDS will be limited. During the selection of a proxy node, we prefer a node with both a fast processor speed and low busy/idle ratio.

To reduce the intrusion detection work on each node, in this study we find a new way to measure the busy/idle ratio of a node. This ratio, to some extent, is reflected by the node's energy consumption rate. If the node is always busy, its energy consumption rate will be fast, and vice versa. Thus the energy consumption rate can denote the node's condition in the network. To simplify the issue, we set only two statuses here: busy and idle. Each node has its own energy consumption rates for the busy and idle statuses. The idle consumption rate should be more stable than the busy one. Because most service nodes provide specific services to users, they have very determined traffic/operation patterns. Thus in this study we can assume the energy consumption rates for both statuses are constant. Suppose that the initial energy is B_0, the current energy at time t is B_t, t_b is the node's total busy time for duration t, t_i is the total idle time for the same duration, R_b is the energy consumption rate for the busy status, and R_i is the consumption rate for the idle status. Based on these factors, the following equations can be deduced:

$$\begin{aligned} R_b \times t_b + R_i \times t_i &= B_0 - B_t \\ t_b + t_i &= t \end{aligned} \tag{9-2}$$

Thus the node's busy/idle ratio C can be represented as:

$$C = \frac{t_b}{t_i} = \frac{\frac{B_0 - B_t}{t} - R_i}{R_b - \frac{B_0 - B_t}{t}} \tag{9-3}$$

9.1.3 Trust

Trust, as we mean it in this thesis, is about the confidence a service node has to delegate its intrusion detection tasks to a specific proxy node. It is important that a proxy node is trustworthy due to the security nature of intrusion detection.

An improper delegation may put the entire system in danger as a compromised proxy node might leave a backdoor to intruders. However, just like in real society, trust is a very subjective metric. A node's trust level may not be the same when measuring it from different angles. In order to be able to compare different nodes' trust levels, the first step is to establish a quantitative trust model.

There are many research results on establishing trust in computer networks [15, 80, 134]. The information sources used to build trust normally are reputation (evidence from observations of previous interactions), delegation or recommendation from a third party. Trust has its own lifecycle. It depends on the procedures of how trust is maintained. A typical lifecycle includes collecting information, evaluating trust, making decision, monitoring, and updating. Trust evaluation and decision-making include risk analysis. There is always a trade-off between risks and benefits. To make the final decision, a pre-defined security policy is essential. For example, it might set a threshold to decide at which level of trust a node can be selected as a proxy node. In the end, the trust must be revocable if a node's trust level is changed or erroneously estimated. It needs to be updated in time.

Because ubiquitous computing is a highly distributed environment, it will be a problem to estimate the trust level for a newly joined node. It is hard to decide where to get recommendations for a totally unknown node. Furthermore, flooding is not an appropriate way to collect recommendations in ubiquitous networks as it may consume more energy.

The paper [74] proposed an intrusion detection method based on the usage of batteries. The principle is that if a node's energy is consumed unusually, it is very likely that the node is under attack. It gave us the idea that the correlations between trust and energy could be measured. In addition, we believe that a node's trust level is also related to its "safe time" (duration without known abnormal activities) in the network. A node with a longer safe time could have a higher trust level. Thus the final estimation of trust is also a coexistence result of multi-factors.

We use T to denote a node's trust level. It can be represented as:

$$T = \frac{f_1(\frac{t_s}{t_e})}{f_2(\frac{B_{t1} - B_{t2}}{(t_2 - t_1) \times (R_b \times S + R_f \times (1 - S))})} \tag{9-4}$$

where $f_1()$ is the function about the safe time and $f_2()$ is the function about the short-term energy consumption rate. In $f_1()$, t_s denotes the node's safe time and t_e is an adjustable threshold. In $f_2()$, B_{t1} and B_{t2} denote the remaining energies at times t_1 and t_2. R_b and R_i are the aforementioned energy consumption rates for the busy and idle statuses, respectively. S represents the node's status between times t_1 and t_2 with $S = 1$ if the node was busy and $S = 0$ otherwise. If the node's short-term energy consumption rate $(B_{t1}\text{-}B_{t2})/(t_2 - t_1)$ is over the conventional value R_b or R_i, the node's trust level will decline. Based on the definitions of $f_1()$ and $f_2()$ listed below, T is confined between 0 and 10 and will be updated every time when the node's status changes.

$$f_1(x) = \begin{cases} x & if\ x \le 10 \\ 10 & otherwise \end{cases} \tag{9-5}$$

$$f_2(x) = \begin{cases} x & if\ x \ge 1 \\ 1 & otherwise \end{cases} \tag{9-6}$$

Because not all attacks can be traced by monitoring the usage of energy, this method is not one hundred percent accurate in reflecting a node's trust level. Other factors such as the node's historical security records and functionalities may give different views on its trustworthiness. More trust related factors will be considered in our future work.

9.2 The protocol

Eventually, to balance these three metrics discussed in section 9.1, we adopt a conditional hybrid metric. Basically, from all the candidates meeting certain prerequisite requirements of a proxy node, the one with the maximum value of

$$M = \frac{E \times T}{C} \tag{9-7}$$

will be chosen. Here, M denotes the hybrid metric, E represents the energy related metric, C is the node's busy/idle ratio, and T is the trust level.

There are two main steps in the selection of a proxy node. The first step is to choose a set of nodes which are capable of taking over the intrusion detection tasks. To become a candidate, a node must have enough processor speed, remaining energy and storage space. The second step is to choose the most

suitable node from the remaining candidates. The decision is made based on the hybrid metric *M*.

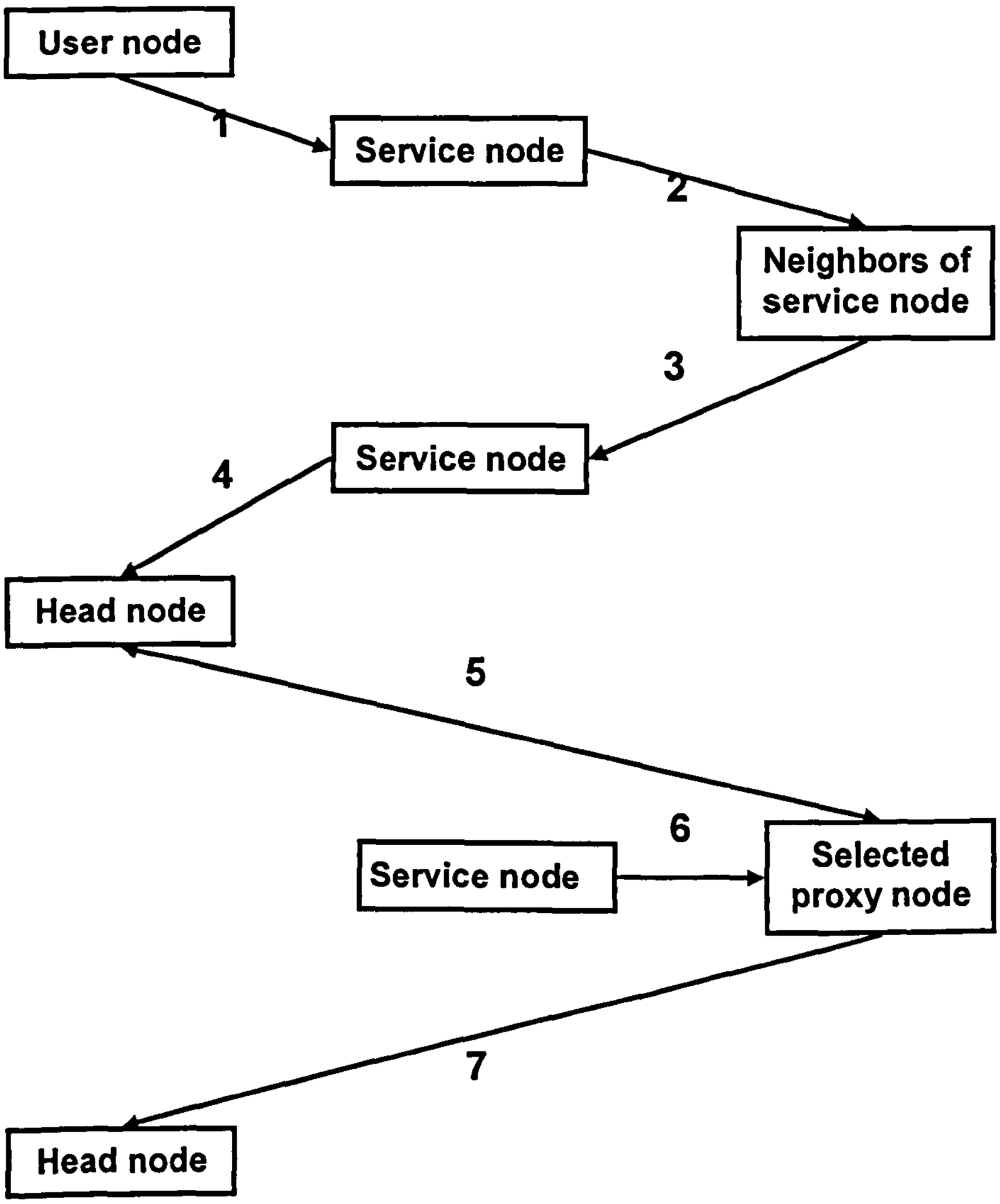

Fig. 9.1 Process for the selection of a proxy node.

The entire process for the selection of a proxy node is depicted in Fig. 9.1, and its steps are explained below:

1. A user requests services from a service node.
2. The service node sends a broadcast message to its neighbours. The TTL (time to live) field of the message is set to 1 (hop) in order to reduce the amount of communication. It contains pre-set minimum requirements on

the three conditional metrics (processor speed, remaining energy and storage space). A threshold value for the next step's hybrid metric is also included to further reduce the communication cost.

3. Every neighbouring node fulfilling the minimum requirements defined in the message received calculates its hybrid metric M, which reflects a combined estimate of the deciding metrics (energy, busy/idle ratio and trust level). If a node's hybrid metric exceeds the threshold set in the message, it sends the result of M to the service node.
4. The service node appoints its proxy node as the one with the highest hybrid metric value among those received, and informs its head node of the decision.
5. After passing an authentication and verification process with the head node, the proxy node retrieves the user's partial profile and the corresponding detection modules from the head node.
6. The service node provides services, while the proxy node updates the user profile based on the event records provided by the service node and monitors anomalies.
7. Before the user leaves the current domain, the updated user profile will be sent back to the head node by the proxy node.

It is worth noticing that in step 5, a verification process is necessary. It is used to prevent a compromised node from winning the position of the proxy node by giving a false hybrid metric value. The winner node has to provide certain details to the head node for verification. Based on equations (9-1), (9-3) and (9-4), a malicious node could modify its remaining energy (B_t) to get a higher hybrid metric value. The modification of other parameters such as the conventional energy consumption rates (R_b and R_i) or safe time (t_s) could be easily identified by the head node to expose the node's malice. As a countermeasure against the illegitimate modification of B_t, we may use a mobile agent to collect the value of B_t directly through the node's I/O interface. To ensure that the mobile agent is not manipulated by the malicious host node, techniques such as obfuscated code [64] and an expiration timer [44] could be used together. The obfuscated code makes the mobile agent code hard to understand in short time so as to prevent its meaningful alteration. The expiration timer requests that a mobile agent must be returned back to its associated head node within certain time interval. If a node

does not return the mobile agent in time, the head node will reject any request for its appointment as a proxy node, and notifies the requesting service node of going back to step 4 for the node with the second highest hybrid metric value. Further actions may also be taken against the suspected node. In this way, even if a node is compromised by an intruder, it cannot gain the access to the intrusion detection modules easily. The implementation of the mobile agent is beyond the scope of this thesis and will be addressed in our future work.

9.3 Experiments and performance analysis

Similar to chapter eight, we use the same simulation environment created by the Georgia Tech Sensor Network Simulator (GTSNetS) [111]. As stated in chapter eight, the simulated network has total 51 nodes in a 120x120m^2 area. There are one head node, ten user nodes and forty service nodes. All the nodes in our simulations are connected and communicate with each other through wireless connections, i.e. in an Ad Hoc pattern. The default routing protocol is DSR [81]. We assume that the signal transmission medium is homogeneous, i.e. fixed α_{11}, α_2 and α_{12} with n=2 in equations (8-1) and (8-2), and all the nodes have the same radio range (*30m*).

9.3.1 Effect of the hybrid metric on network lifetime

The first experiment is to examine the effect of the hybrid metric defined in equation (9-7). We created three scenarios. The first scenario demonstrates our previous design, i.e. all event records are sent to and processed at the head node. The distances between the service nodes and the head node are calculated based on their randomly deployed positions. In the second scenario, the service nodes process the event records locally without any help from proxy nodes. Thus the communication-related energy consumption can be reduced. In the last scenario, once a service node has been activated, all the nodes within its radio range, including itself, will be examined against the hybrid metric. The node with the highest hybrid metric value will be chosen as its proxy node. The event records will be sent to the proxy node and processed there.

In all these three scenarios, the communication-related energy is calculated based on equations (8-1) and (8-2). The energy consumed on implementing intrusion detection tasks is shared between the service nodes and their proxies. We assume that the energy consumed at a service node is proportional to the length of an event record it generates, and the energy consumed by its proxy node is a fixed cost (*5mJ*) for the record reception and processing. To get the hybrid metric, we need to know the nodes' conventional energy consumption rates (R_b and R_i) in prior. They have been calculated through a training process. We pre-ran the simulation several times with different active user nodes. A node's energy consumption rate at the busy status (R_b) is calculated by dividing the energy consumed during its active sessions by the total busy time. And similarly, R_i is the result of dividing the energy loss during other times by the total idle time. In the end we use the means as the conventional rates. The simulation stops after the network is dead (i.e., half of the service nodes are battery exhausted).

We tested each scenario with different user nodes and took the mean values as the final results. Fig. 9.2 shows their differences. As with the previous experiments, the horizontal axis in the figure denotes the number of dead nodes, and the vertical axis denotes their death time. At the node number equals to 20, the network is dead. Not surprisingly, with our hybrid metric, the system has the best performance comparing with simply using a head node or service node itself to process the event records. The impact of the hybrid metric gradually improves as the simulation proceeds. The average node lifetime is increased from 4284.445s and 4650.8s to 5116.775s (increased 19.42% and 10.02%, respectively). The network lifetime is extended on average from 5862.9s and 6048.4s to 7165.3s (increased 22.21% and 18.47%, respectively).

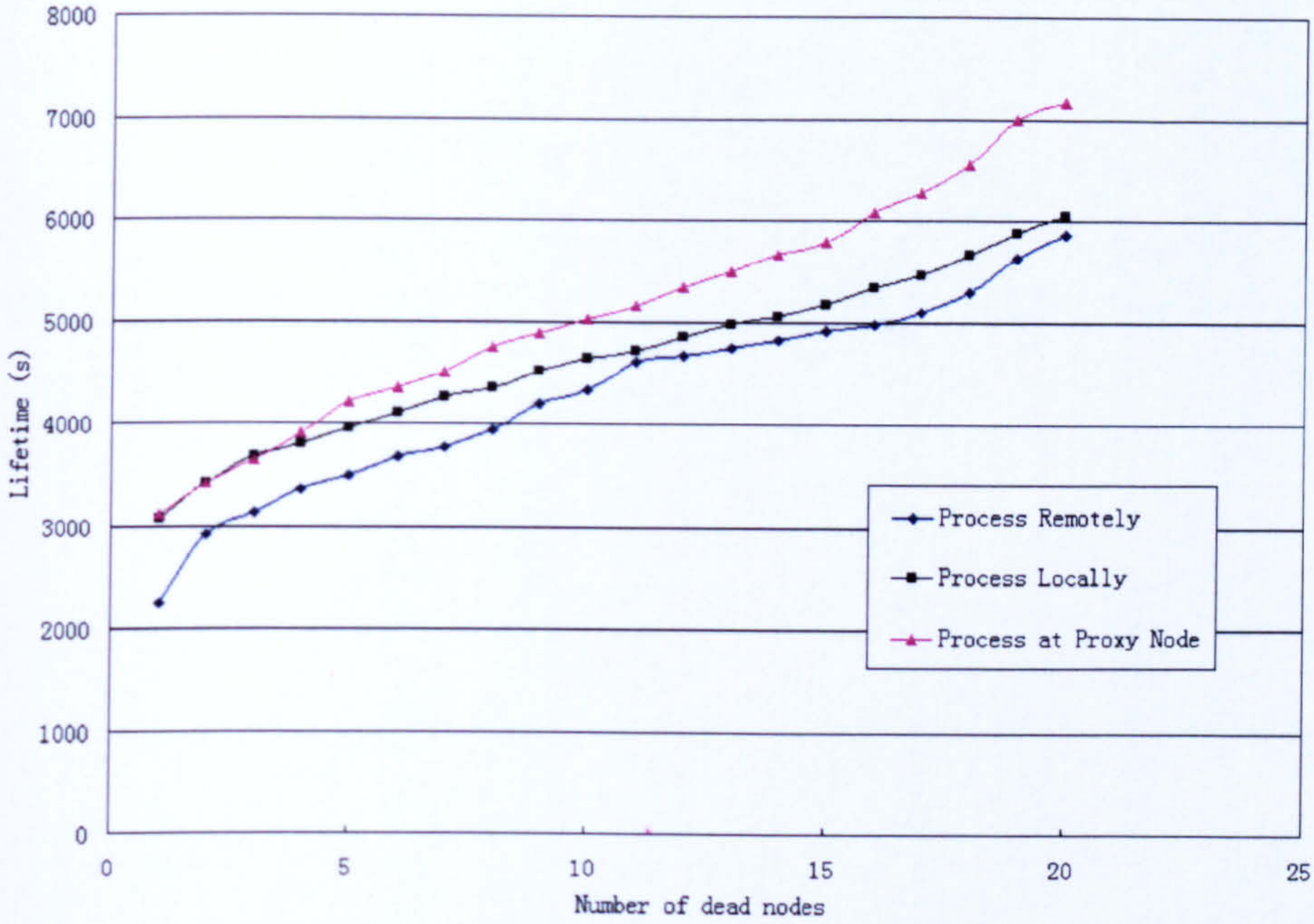

Fig. 9.2 Impact of the proposed hybrid metric.

9.3.2 Enhanced security policy under attacks

In the second experiment we monitor the reaction of the hybrid metric to attacks and see how the system's security can be enhanced. We simulated two types of anomalies or attacks in this experiment. The first type is a denial-of-service attack, which is generated by deliberately occupying the CPU time of a service node. The second type is a SYN flood attack. In both cases the attacker randomly picks a victim from the first half of the service nodes. After each attacking session, the safe time (t_s) of the victim node will be reset to zero. Ideally, the attacked node should have a lower trust level and less chance of being selected as a proxy node in short time. In this way, the system is enforced with stronger security assurance.

Fig. 9.3 and 9.4 show the different proxy node selection distributions before and under attacks. The horizontal axis denotes the node number and the vertical axis denotes how many times a node has been selected as a proxy node during the simulation. Among these nodes, node 1 is the head node, nodes 2 to 11 are the user nodes, and the rest are service nodes. As we mentioned earlier, the first half

of the service nodes (numbers 12-31) are the victims of the attacks. We can see that before the attacks, the selection distributions are similar for the different groups of nodes. Afterwards, the chance for the victim group to be chosen as proxy nodes has fallen down significantly and the others have increased their chances accordingly. Specifically, the average selection time for the victim group of nodes is down from 5.05 to 2.5 and the average lost selection percentage is 46.3%.

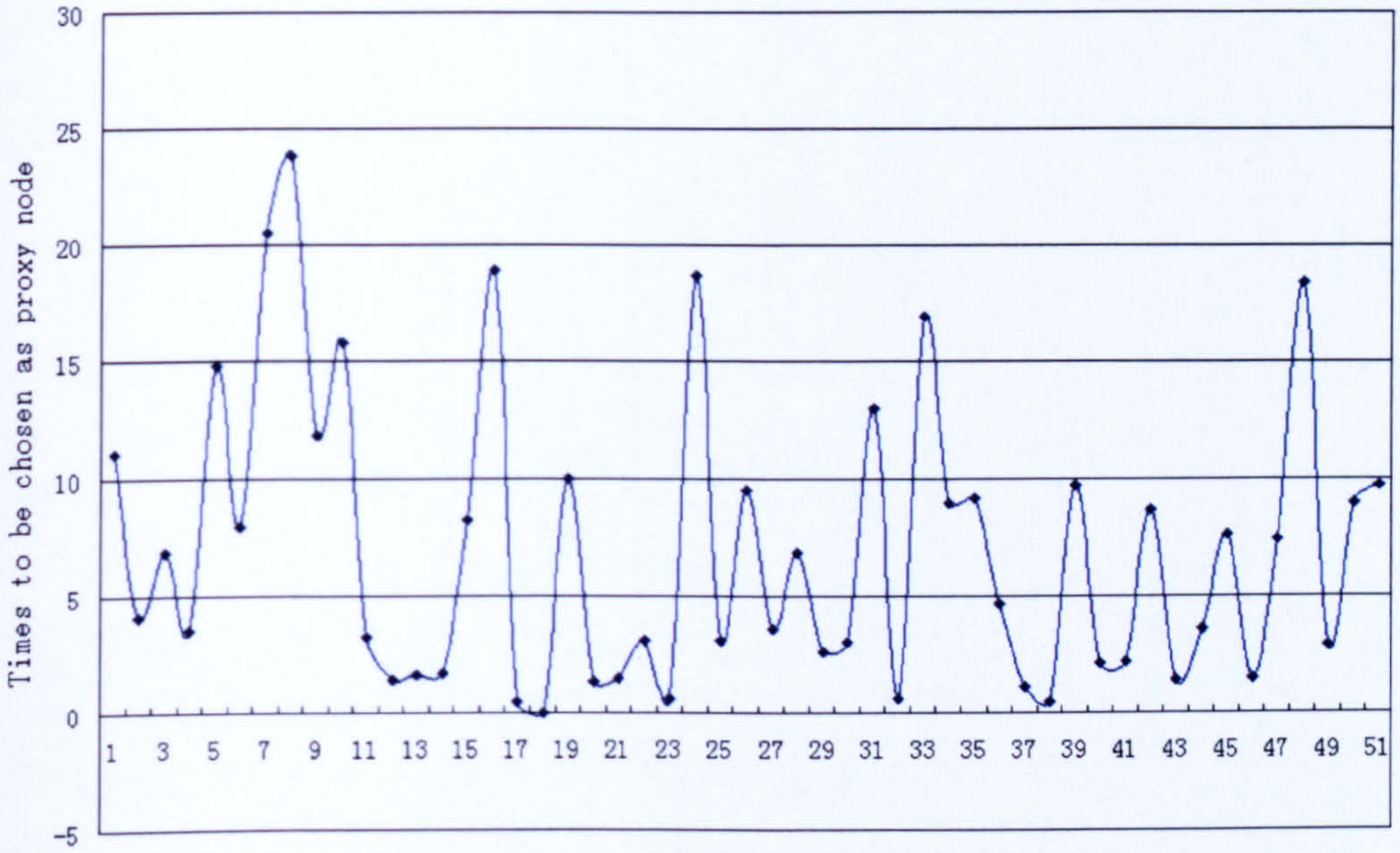

Fig. 9.3 Proxy selection distributions before attacks.

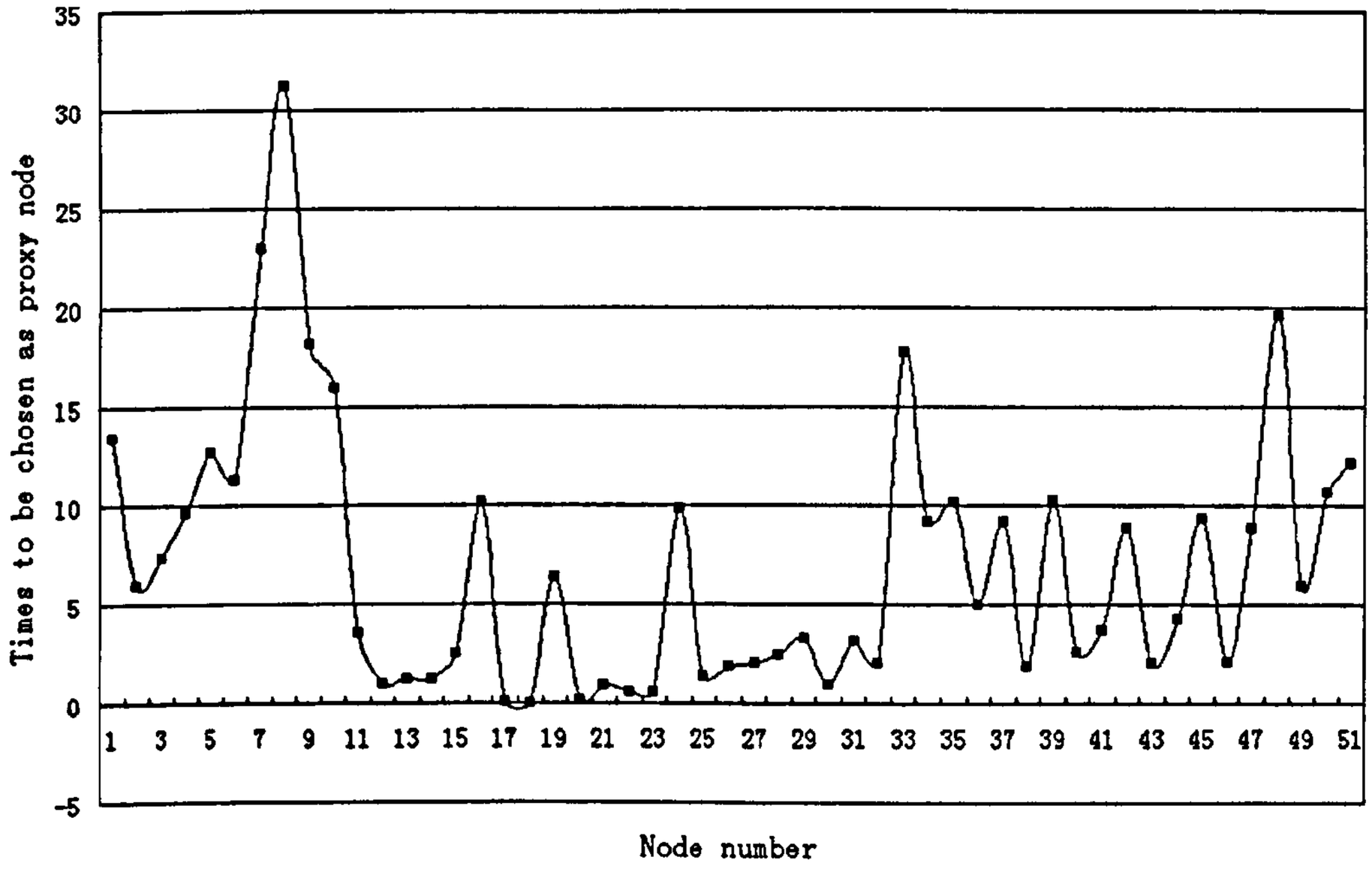

Fig. 9.4 Proxy selection distributions under attacks.

9.4 Summary

SUIDS is an anomaly-based intrusion detection system proposed for ubiquitous computing environments. It meets the special requirements of ubiquitous networks by taking resource constraints into account. In this chapter, we analyzed the requirements on resources in SUIDS and proposed a profile splitting technique to achieve resource efficiency. Instead of sending event records to a fixed node for processing, a proxy node is selected based on the availability of its resources. Three deciding resource factors have been considered: energy, computing ability and trust. In order to balance these three factors, we proposed a novel conditional hybrid metric. Our experiment results show that by applying the hybrid metric, SUIDS achieves better performance in terms of resource-efficiency, and also its security assurance is beneficially enhanced.

CHAPTER TEN

SUIDS EVALUATION

In this chapter, we review the requirements of intrusion detection systems in ubiquitous computing and evaluate the performance of SUIDS against them. A successful intrusion detection system in ubiquitous computing must have the following five features: real-time detection, scalability and adaptability, full coverage, resource efficiency, and detection effectiveness. Comparing with existing solutions, SUIDS addressed all these issues from the start of its design. Specifically, it achieves real-time detection by giving mobility to its detection modules. The classification of network nodes and usage of lightweight agents make SUIDS scalable and adaptable; SUIDS considers capacity constrained nodes by adopting proxies; a novel hybrid metric balances the system resources; and in the meantime, SUIDS achieves a high detection rate while keeping its false alarm rate low.

10.1 Requirements on IDSs in ubiquitous computing

Before listing the requirements on an IDS, we first look at what need to be protected in ubiquitous networks. Conventionally, IDSs are used to protect computers and computer networks against any malicious activities that could compromise the integrity, confidentiality, or availability of the network or information sources. According to the definition of ubiquitous computing, computer embedded devices will eventually spread throughout physical environments. For example, our TV, refrigerator, and even door lock might be equipped with computer processors and connected together. The question is whether an IDS has to protect all the nodes within a network, or we can just leave

some of them alone. Actually the answer depends on the network/system's security policy. The same device might need protection in certain scenarios but the other way round in other situations. There is always a trade-off between the level of security and the usage of resources needed for the security protection.

Imagine there is a smart refrigerator, which is able to notify a user about what kind of food and how much is left inside and what their 'use by' dates are. The user might be allowed to set the temperature he/she wants the refrigerator to keep and decide the period of refilling the refrigerator with fresh food, to make sure that the food inside the refrigerator is always adequate and healthy. All these enquiries and settings could be completed through wireless connections by using the user's PDA. Some people might feel that there is nothing to do with security and intrusion detection. Then think about what will happen if someone modifies the data about the temperature the refrigerator should keep and thus makes the food become inedible, or orders the refrigerator keeping food which is already beyond the 'use by' date. Surprisingly we can conclude that in ubiquitous computing a malicious user may threaten not only our information resources, but even our finance and health by simply controlling a smart refrigerator.

What makes the security situation in ubiquitous computing sound worse? If we look back to the developing history of computers and intrusion detection systems, we might find the answer. Actually the scope of intrusion detection was always growing with the popularization of computers. In the early period the only benefit a hacker could obtain from attacking was just making free phone calls. But today, they can certainly benefit much more since computers are applied to various areas. It is predictable that in ubiquitous networks hackers can do even more as long as computer embedded devices are manipulatable. It is ironic that we introduce computers into our lives to make things easier, but at the same time give hackers opportunities to take advantage of it.

As computers become ubiquitous, intrusion detection will be closely connected with our daily lives. For example, an IDS may need to monitor who is using a smart refrigerator, for how long and how often, and who is trying to open electric doors. The border between intrusion detection and user surveillance will become obscure. It is difficult to distinguish them as totally separated. The issue of user surveillance is related to privacy protection. It cannot be solved by only technical means. A proper security policy and privacy policy are both needed.

Such policy issues are beyond the scope of our research.

The above discussion clearly indicates that the evolving infrastructure of networks to support ubiquitous computing requires the development of a new generation of resource-efficient IDSs to provide appropriate protections for ubiquitous computing environments. Such an IDS must minimise the use of devices or network nodes with limited resources (e.g. energy and communication resources) for intrusion detection while being able to achieve high detecting efficiency (e.g. a high hit rate and a low false alarm rate). The new IDS needs an appropriate system architecture and strategy to make it flexible and scalable.

There are two key requirements for all IDSs: effectiveness and efficiency. The effectiveness refers to that the IDSs must be able to distinguish malicious actions from normal actions correctly. Both false positive (label normal activities as malicious) and false negative (overlook malicious activities) decisions are undesirable and must be kept under certain level. The efficiency means that an IDS must run in a cost-effective way. Excessive overhead introduced by the IDS on CPU usage, network resources and storage space confines the wide deployment of the IDS. The implementation of the IDS should not disturb existing systems doing their normal activities.

Keeping these two basic requirements in mind, we now expand in detail what is exactly required for IDSs, especially considering the impact of the characteristics of ubiquitous computing on intrusion detection.

- Real-time detection: An IDS must run continuously, or at least periodically, to detect intrusions and make the corresponding responses. A delayed monitoring may cause crucial losses and give intruders chances to hide or remove their trails. As we explained earlier, in ubiquitous computing, the consequence of a successful attack could harm physical environments, so real-time detection becomes especially important.
- Scalability and Adaptability: An IDS should be scalable. The IDS in ubiquitous computing must be able to cope with hundreds, or even thousands, of network nodes. An IDS must be adaptable as well. System and user behaviours are changing over time. The topologies of networks are also varying. In ubiquitous computing, the situation is even more complicated as some hosts are capable of mobility. The IDS must be able to adapt to these changes.

- Full coverage: In ubiquitous computing, an IDS needs to consider those nodes that are incapable of implementing the IDS by themselves. Effectively deploying the IDS in an environment with such a diverse range of devices/nodes is a big challenge. An IDS should be organized in a distributed manner. Balancing the computing load and diagnostic burden of intrusion detection among network nodes can increase the network/system's fault-tolerance, scalability and security protection coverage.
- Resource-efficiency: An IDS should require as little system resource usage as possible to alleviate extra burdens on CPU usage, network overhead, storage space and battery consumption. In ubiquitous computing, many devices may have very small physical sizes to achieve their unaware/invisible deployment. Although manufactures keep working on enhancing the capacity of their products, many appliances/devices will still face limitations on system resources, especially for those battery-powered.
- Detection effectiveness: An IDS must be able to detect malicious activities effectively. It must keep both false positive and false negative alarm rates under acceptable levels.
- Low administration burden: Because ubiquitous computing is related to people's daily lives, an IDS must keep the administration burden low. Normal users cannot be expected to have many security expertises.

10.2 Evaluation of SUIDS

10.2.1 System architecture

Among the five requirements stated earlier, real-time detection, scalability and adaptability, and full coverage are related to system architectures. We have compared SUIDS with other IDSs mentioned in chapter four with regard to these three requirements, and list the comparison results in Table 10.1. For conventional hierarchically organized IDSs such as GrIDS and EMERLAND, they were proposed for static wired networks and do not fit for topology-varying network

environments such as those for ubiquitous computing. Newly emerged IDSs which were proposed for mobile ad hoc networks overcame this problem by using a cooperative architecture or software agents. However, they did not consider those nodes that lack abilities to implement an IDS module independently, because current ad hoc networks mainly utilize relatively powerful devices such a laptop or PDA. SUIDS is featured by a distributed auditing scheme followed by a lightweight intrusion detection analysis. It can adapt intrusion detection tasks to fit the operational characteristics of service and user nodes in a network, process event records in real-time, and use proxy nodes to balance the network resources for the intrusion detection coverage of resource poor small devices/nodes. Thus SUIDS is the only intrusion detection system that fulfils all these three requirements.

Table 10.1 Comparing SUIDS with other IDSs in respect of system architectures.

Introduced IDSs	Real-time detection	Scalability and adaptability	Full coverage
SUIDS	Yes. User profiles and detection modules are mobile and lightweight. Anomalies are detected in an online fashion.	Yes. The layered structure, network node classification, and service-oriented user-centric design help effectively organise the entire system.	Yes. Use proxy nodes to balance network resources. Resource constrained nodes can delegate intrusion detection tasks to proxy nodes.
DIDS	Delayed as event reports need to be sent and processed at the central manager.	Only work in IP-based environments.	No, require that all hosts be C2 or higher rated computers [153].
GrIDS	Need to wait a detection window before the aggregation of network activities.	No, proposed for conventional static wired networks.	Yes, monitor connections and do not need all nodes' participation.
EMERLAND	Yes, each monitor may own anomaly and misuse detectors.	No, its subscription mechanism introduces high network overload.	No consideration, proposed for powerful PC-based networks.

IDSs for Ad Hoc	Yes, each IDS agent has a local detection engine.	Yes, totally distributed.	No consideration, every node needs to run an IDS agent independently.
Indra	May be delayed, depending on the size of a network.	No, introduce high trust management overload.	No consideration.
AAFID	Yes, network latency reduced as its IDS agents operate locally.	Yes, but still rely on a central entity.	No consideration, each host contains a transceiver, filter and any number of agents.
Sparta	Depend on network sizes and mobile agent roaming patterns.	No, its directory service is not suitable for a large-scale network.	No consideration, its complex mobile agent is too heavy for small nodes.
Function based MA	Depend on network sizes and mobile agent roaming patterns.	Yes, fully distributed system architecture.	No consideration, but the requirement for running an IDS agent is much lowered.

10.2.2 Resource efficiency

Resources are crucial in ubiquitous computing. Any applications designed for ubiquitous computing should set resource-efficiency as one of the main objectives. However, most existing IDSs did not consider the resource constraints in their design. The reason is that they were proposed for either wired networks or ad hoc networks (laptop or PDA based). The requirements on resources in these environments are not as strict as those in ubiquitous computing. Current resource efficient techniques which were proposed as complements of existing IDSs do not fit for ubiquitous computing. Table 10.2 summarises their drawbacks. The resource efficiency issue must be carefully considered before the implementation of an IDS in ubiquitous computing environments.

SUIDS uses a profile splitting technique to achieve resource efficiency. Instead of sending event records to a fixed node for processing, a proxy node is selected dynamically based on the availability of network resources. Three deciding factors have been considered for the proxy node selection: energy, computing ability and trust. In order to balance these three factors, we proposed a

novel conditional hybrid metric. As demonstrated in chapter nine, our experiment results showed that by utilizing the hybrid metric, SUIDS achieves better performance in terms of resource-efficiency. The average node lifetime in our experiments was increased by at least 10% and the network lifetime was extended on average by at least 18%. Besides, because SUIDS took the nodes' trust level into account before delegating intrusion detection tasks, its security assurance is beneficially enhanced. The methodology used in SUIDS could benefit the further development of IDSs for ubiquitous computing networks in the future.

Table 10.2 Drawbacks of current resource efficient solutions for IDSs.

Resource efficient solutions	**Drawbacks**
Choose a cluster head for implementing an intrusion detection module at any given time.	Require all nodes pre-install IDS modules and be able to carry out intrusion detection tasks independently.
Quantify damage costs based on an intrusion's type and its target. An optimized model reduces detection costs by intelligently rearranging detection rules.	The quantification of attack costs is complicated and costly. It lacks a common standard, as in different scenarios the same attack may cause unequal losses.
Address NIDSs overload problems by running the most crucial event in front of others for performance monitoring.	Cannot solve the overload problems thoroughly without any help from HIDSs.
Apply an adaptive response mechanism by balancing parameters such as a false alarm rate, detection confidence and damage cost.	Need a quantification process similar to the second method above.

10.2.3 Detection effectiveness

In this thesis we presented two detection techniques, a string-based approach and a chi-square statistic test, in chapters 6 and 7, respectively. We now compare

their performances based on the experiment results discussed in these two chapters. Table 10.3 summarises the comparison outcomes. For detection effectiveness, the experiment results are close. The string-based approach has a slightly better hit rate, while the chi-square statistic test has a lower false alarm rate. Both of them are lightweight and can detect anomalies in real-time. A user profile used for the string-based approach (with a string length of 80) is bigger than that for the chi-square statistic test. Considering that we only set ten service nodes in the experiments, the difference between them will definitely grow in a larger network. Besides, because the string-based approach is based on a cumulated result, it needs an extra process to identify a malicious event record. Thus we think the chi-square statistic test is a better solution in general.

Table 10.3 Comparison of two detection methods.

	String-based approach	**Chi-square statistic test**
False Alarm Rate (R_{fp})	3.75%	3.27%
Hit Rate by Records (R_h)	95.03%	94.09%
Hit Rate by Session	100%	100%
Real-time detection	Yes	Yes
User profile size (KB)	6.22	5.25

In conclusion, SUIDS adopts a layered and distributed system architecture, which is seamlessly embedded into the ubiquitous computing environments. By categorizing the system nodes into three major groups, SUIDS is more scalable and adaptable in order to fit for various network scenarios. SUIDS has a novel user-centric design and service-oriented detection method. By giving the mobility to detection modules, SUIDS is able to react to malicious activities in real-time. It detects anomalies at the service level rather than relying on a one-sided network

layer. SUIDS also equips a new resource-sensitive scheme, including protocols and strategies. By allowing the delegation of intrusion detection tasks to proxy nodes, it provides satisfactory intrusion detection service coverage to those nodes that are incapable of running IDS independently. A novel hybrid metric based algorithm is used in SUIDS in order to balance the system resources such as CPU usage, network overhead, storage space, and energy consumption. This hybrid metric can measure these factors together by exploiting their hidden correlations. A node's trustworthiness is also considered in this hybrid metric to enhance the system's security policy. The effectiveness of SUIDS is reflected by its high hit rates on anomalies and low false alarm rates. Its efficiency is shown on the deducted energy consumptions. All these novelties and characteristics make SUIDS well fit for ubiquitous computing environments.

10.3 Summary

System evaluation is an important consideration in any system development. In previous chapters, we presented the system architecture design, detection methods, and resource-efficient solutions of SUIDS. This chapter evaluated the entire system in relation to the requirements stated in section 10.1. The evaluation demonstrated the novelty of SUIDS in its architecture design. To the best of our knowledge, SUIDS is the first intrusion detection system that took the special requirements of ubiquitous computing into account during its design. It adopted proxy nodes in intrusion detection and used a novel hybrid metric to balance multiple system resources such as energy, computing ability, and trust information. The detection algorithms of SUIDS were tested with a number of parameters such as a hit rate, false alarm rate, and user profile size. As the test results demonstrated, SUIDS provides a robust and resource-efficient protection for ubiquitous computing networks.

CHAPTER ELEVEN

CONCLUSIONS AND FUTURE WORK

This chapter revisits the themes recurrent in this thesis and details future work. The notion of ubiquitous computing was introduced as a prospective view about the future usage of computers. Smaller and cheaper computer chips will enable us to embed computing ability into any appliances. Existing IDSs have several weaknesses that hinder their direct application to ubiquitous networks. These shortcomings are caused by their lack of considerations about the heterogeneity, flexibility and resource constraints of ubiquitous networks. As demonstrated earlier, to overcome these problems, we proposed a novel service-oriented and user-centric intrusion detection system – SUIDS. SUIDS is an adaptive and resource-efficient intrusion detection system with a novel service-oriented auditing mechanism and flexible user-centric design. By working together with service-oriented agents, SUIDS can reliably and effectively detect malicious activities of inside users. SUIDS comprises the following main components: a reliable auditing mechanism, a resource-efficient intrusion detection scheme, and a flexible system architecture. Our future work will focus on the further examination of SUIDS and the refinement of its models.

11.1 Conclusions

This thesis first introduced the history of computer networks. For many years, computers were supposed to stand alone, run programs and provide computing resources for local usage only. This situation changed with the advent of ARPANet in the late 1960's. A set of computers were connected together in order to allow remote access to computer resources. Since then, millions of computers

joined the network forming the biggest computer society - Internet. By enabling us to shop, work and study remotely, the Internet changes our daily lives in many ways.

Soon, with the continuous growth and development of computer and network technologies, we will enter the next stage of information era – ubiquitous computing. The concept of ubiquitous computing was introduced as a prospective view about the future usage of computers. Smaller and cheaper computer chips will enable us to embed computing ability into any appliances, e.g. a cup, lighter, and even a piece of paper. People's daily activities will be closely connected with computers and beneficially become ever convenient. For example, in ubiquitous networks, one can open a door by simply sending an order to the electric door lock from his/her PDA, or read news on a computer embedded "e-paper" with the content updated through wireless connections.

However, the great features of ubiquitous computing inevitably expose its inherent vulnerabilities. The convenience brought by ubiquitous computing could also be taken advantage of by intruders. It makes things too easy for malicious people to build a system to spy on others. For example, an intruder may compromise the integrity and confidentiality of an information system by using a stolen ID to modify or access valuable information, or compromise the availability of an information system by possessing the system resources in order to interfere with authorised users' normal access. Like any other computer systems, one of the main prerequisites for the wide adoption of a ubiquitous network is security. The network has to be properly secured so that it can be relied upon.

IDSs are widely used to protect computer networks. In computer security, intrusions are defined as any malicious activities that could compromise the integrity, confidentiality, or availability of networks and information sources. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Moreover, an effective IDS can even serve as a deterrent, acting to prevent intrusions. As a second line of defence, IDSs play an important role in computer security, especially in the fight against attacks launched inside networks.

Two principal classifications of IDSs have been explained in this thesis. According to the detection methods used, IDSs can be divided into

signature-based detection and anomaly-based detection. The signature-based detection compares audit data with the knowledge accumulated about specific known attacks and system vulnerabilities. The anomaly-based detection builds a reference model of the usual behaviour of the system being monitored and looks for deviations from the normal usage. According to the locations of audit sources, IDSs can also be categorized as host-based IDSs and network-based IDSs. Host-based IDSs audit data mainly from local operating systems, e.g. system log files, and network-based IDSs audit network packets between nodes or the Simple Network Management Protocol information.

Although the research in intrusion detection started decades ago, its application to ubiquitous computing is new. This thesis provided a critical survey on existing solutions. As concluded, they do not fulfil the special requirements of ubiquitous computing in respect of resource-efficiency and system architecture. Specifically, an IDS in ubiquitous computing should not require to transmit or process a large amount of audit data or attack signatures; a centralized detection scheme should be replaced by a distributed or cooperative system architecture; host-based and network-based approaches should work together to provide all-sided protection. Within our knowledge, there is no IDS yet, which has been particularly proposed to meet these special requirements of ubiquitous computing.

As a solution to address this issue, we proposed an adaptive and resource-efficient IDS with a novel service-oriented auditing mechanism and flexible user-centric design – SUIDS. SUIDS handles the heterogeneity issue of ubiquitous computing networks by classifying network nodes into three major categories (head nodes, service nodes, and user nodes) and integrating intrusion detection with service specific knowledge. SUIDS is a distributed and dynamically deployed system based on this classification.

Unlike existing network-based IDSs, SUIDS integrates service specific knowledge with intrusion detection and thus focuses on the service level instead of burdensome packet analysis. Agents on service nodes monitor system information across the system layers, e.g. from the network layer such as an open port to the application layer such as a device operation. The information eventually converges to the service level. In this way, the SUIDS detection modules on head nodes can reliably and effectively detect malicious activities of inside users and only need to analyze event records instead of a bundle of packets.

Two anomaly-based detection methods have been tested with SUIDS. The first one is a string-based approach. Some IDSs use a time interval to determine a detection window, i.e., each event only makes effect during a certain period. Because SUIDS is a distributed and mobile system, the use of a time-based detection window will introduce synchronisation issues and make the system more complicated. Hence in the string-based approach, a 'string' is used to indicate a user's short-term behaviour in an online fashion. For example, if the last 100 printing operations can effectively represent a user's short-term behaviour regarding his usage of the printer, a string with the length of 100 will be set to follow the printing probability distributions in the user profile. Each character of the string represents one of his/her historical operations. Every time when a new record comes, the earliest record will be discarded. The length of the string is a variable, depending on the system requirements. Generally, a longer string could have a lower false positive rate but with the possibility of increasing the false negative rate at the same time and consuming more system resources.

The second detection method is based on a chi-square statistic test. Instead of using a string, an exponentially weighted moving average (EWMA) technique is used to smooth out observation values for the variables being tracked. It applies a smoothing constant in a user profile to represent the user's short-term behaviour in real-time. In this way, the most recent and past records have different weight indexes. The observation reflects the 'most recent past' characteristics of the variables. The deviations between a user's short-term and long-term behaviours are measured by using a chi-square statistic test. Comparing with the string-based approach, this method can measure not only the probability distributions of the variables, but also their occurrence patterns and hidden correlations. As a result, SUIDS achieves real-time intrusion detection in ubiquitous networks with a lightweight and adaptable detection model.

According to the definition of ubiquitous computing, many embedded computer chips must be physically small in order to achieve unaware deployment. Inevitably, they will have limited system resources such as energy supplies and storage spaces. Hence SUIDS has to give special concerns over the issue of resource-efficiency. To further improve the performance of SUIDS, we presented a comprehensive analysis of energy consumed in SUIDS. The energy consumptions in SUIDS are categorized into two parts: computing-related and

communication-related. The computing-related part can be reduced by taking advantage of head nodes' unlimited power supplies; and the communication-related part can be reduced by having user profiles distributed and implementing the SUIDS detection modules locally. To balance these two parts, this thesis proposed a profile splitting technique and a new hybrid metric. Instead of sending event records to a fixed node for processing, a proxy node is selected based on the calculation of the hybrid metric. The hybrid metric considered three energy-related factors on a node: its transmission power, residual energy, and energy consumption rate. Our experiments indicated that this method successfully distributed the energy consumptions of intrusion detection among network nodes and extended the network lifetime.

Based on the above results, we extended our work in order to take other system resources into account to enhance the resource efficiency of SUIDS. In the latest scheme, four key resources have been considered during the selection of a proxy node: its energy, storage space, processor speed (busy/idle ratio), and trust. Among them, the energy, storage space and processor speed are quantified metrics. It is possible to compare them directly by knowing their numerical values. In contrast, an extra process is used to evaluate the trust level of the node. This process calculates the node's trust level based on its energy consumption pattern and its safe time in the system. In order to balance these four resources together, we proposed a new conditional hybrid metric. This metric effectively exploited the hidden correlations among the resources.

System evaluation is an important consideration in any system development. In the previous chapter, we reviewed the requirements on IDSs in ubiquitous computing and evaluated the performance of SUIDS against them. A successful IDS in ubiquitous computing must have the following five features: real-time detection, scalability and adaptability, full coverage, resource efficiency, and detection effectiveness. Comparing with existing solutions, SUIDS addressed all these issues from the start of its design. Specifically, it achieves real-time detection by giving mobility to its detection modules. The classification of network nodes and usage of lightweight detection agents make it scalable and adaptable. SUIDS offers the intrusion detection coverage of capacity constrained nodes by adopting proxies. The novel hybrid metric of SUIDS balances multiple system resources to achieve optimal efficiency. Moreover, SUIDS can achieve

high detection effectiveness while keeping its false alarm rate low.

In conclusion, SUIDS is the first intrusion detection system proposed for ubiquitous computing environments. It keeps the special requirements of ubiquitous computing in mind throughout its design and implementation. SUIDS adopts a layered and distributed system architecture, a novel user-centric design and service-oriented detection method, a new resource-sensitive scheme, including protocols and strategies, and a novel hybrid metric based algorithm. These novel methods and techniques used in SUIDS set a new direction for future research and development. As the experiment results demonstrated, SUIDS is able to provide a robust and resource-efficient protection for ubiquitous computing networks. It ensures the feasibility of intrusion detection in ubiquitous computing in the first place. This work has been recognised by many international academic organizations. In total, eight papers have been published and one more paper is under review.

11.2 Future work

This thesis has posited the weaknesses of current intrusion detection solutions, and the requirements for a new generation of intrusion detection that protects ubiquitous computing networks in a resource-efficient way. As presented in this thesis, SUIDS provides a resource-efficient, scalable, and effective approach.

Future work includes several main directions:

- Refinement of the SUIDS model and detection techniques for improved defence against attacks. For example, exploring more complex algorithms such as neural networks may further reduce the false alarm rate of SUIDS and increase its detection effectiveness.
- Improvement of resource measurements for higher accuracy. For example, the computing-related energy is referred to the energy used to implement the SUIDS intrusion detection modules. It is mainly dedicated to monitor network statuses and user activities, execute the intrusion detection algorithms, maintain and update user profiles. This thesis used a simple model to calculate this part of energy consumption, assuming it is proportional to the number of event records. A more detailed definition

and simulation model may increase the accuracy of the energy consumption measurement.

- Extension of the trust measurement model. Currently the trust measurement model in SUIDS considers only two factors: energy consumption pattern and safety time. It works as a draft model rather than the final solution. Trust management itself is an area where many attentions have been attracted to. We pointed out one possible solution and a refined model could be expected to appear in future research.
- Creation of a prototype of SUIDS in a laboratory for further examination. This thesis has demonstrated the effectiveness and efficiency of SUIDS in a simulated environment by using GTSNetS. Implementing a small ubiquitous computing network such as a smart home will help us to explore the applicability of SUIDS and get more convincing results. For example, we can investigate a user's behaviour and monitor the resource consumption of SUIDS in runtime.
- Cooperation with other information security countermeasures and non-security factors such as law enforcement and privacy protection strategy. An intrusion detection system alone cannot solve all the security issues. It has to work closely with other defence mechanisms such as cryptographic support, security policy enforcement, and access control. In different application scenarios, system conditions and requirements are not the same. SUIDS must take other available security countermeasures into account in its future utilizations.

11.3 Summary

Nowadays our economy relies heavily on networked computer information systems for commerce, communications, energy distribution and transportation, as well as a host of other critical activities. One of the key security issues requiring urgent attentions is about how to protect system resources against malicious activities. A recent FBI survey suggests that 44% organizations had experienced intrusions from within their organization. The average cost of a successful attack by a malicious insider is much greater than the cost of an external attack. It

emphasizes the needs for one type of security tool – Intrusion Detection Systems. The SUIDS project has highlighted the problems of current intrusion detection solutions in ubiquitous computing environments and provided a resource-efficient solution as an important first step toward meeting the special requirements of ubiquitous computing networks.

REFERENCES

[1] Abowd, G., and Mynatt, E., Charting past, present, and future research in ubiquitous computing. ACM Transactions on Computer-Human Interaction, v 7, n 1, Mar. 2000, pp. 29-58.

[2] Aime, M., Calandriello, G., and Lioy, A., Dependability in wireless networks: Can we rely on WiFi? IEEE Security and Privacy, v 5, n 1, Jan./Feb., 2007, pp. 23-9.

[3] Alampalayam, S., and Kumar, A., An adaptive and predictive security model for mobile ad hoc networks. Wireless Personal Communications, v 29, n 3-4, Jun. 2004, pp. 263-81.

[4] Axelsson, S., Intrusion detection systems: A taxonomy and survey. Technical Report 99-15, Department of Computer Engineering, Chalmers University of Technology, Sweden, Mar. 2000.

[5] Bai, Y., and Kobayashi, H., Intrusion detection systems: technology and development. Proceedings of 17th International Conference on Advanced Information Networking and Applications (AINA 2003), Xi'an, China, Mar. 2003, pp. 710-5.

[6] Balachandran, A., Voelker, G., and Bahl, P., Wireless hotspots: current challenges and future directions. Mobile Networks and Applications, v 10, n 3, Jun. 2005, pp. 265-74.

[7] Balasubramaniyan, J., Garcia-Fernandez, J., Isacoff, D., Spafford, E., and Zamboni, D., An architecture for intrusion detection using autonomous agents. Proceedings of 14th Annual Computer Security Applications Conference, Scottsdale, USA, 1998, pp. 13-24.

[8] Bernardes, M., and dos Santos Moreira, E., Implementation of an intrusion detection system based on mobile agent. Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems, Limerick, Ireland, 2000, pp. 158-64.

[9] Bhuse, V., and Gupta, A., Anomaly intrusion detection in wireless sensor networks. Journal of High Speed Networks, v 15, n 1, 2006, pp. 33-51.

[10] Bonifacio, J., Cansian, A., de Carvalho, A., and Moreira, E., Neural networks applied in intrusion detection systems. Proceedings of IEEE International Conference on Neural Networks, v 1, IEEE World Congress on Computational Intelligence, Anchorage, USA, 1998, pp. 205-10.

[11] Bononi, L., and Tacconi, C., A wireless intrusion detection system for secure clustering and routing in ad hoc networks. Proceedings of 9th International Conference on Information Security, ISC 2006, LNCS, v 4176, Samos, Greece, 2006, p 398-414.

[12] Briscoe, R., The implications of pervasive computing on network design. BT Technology Journal, v 22, n 3, Jul. 2004, pp. 170-90.

[13] Brutch, P., and Ko, C., Challenges in intrusion detection for wireless ad-hoc networks. Proceedings of Symposium on Applications and the Internet Workshops (SAINT 2003 Workshops), Orlando, USA, 2003, pp. 368-73.

[14] Bull, P., Limb, R., and Payne, R., Pervasive home environments. BT Technology Journal, v 22, n 3, Jul. 2004, p 65-72.

[15] Cahill, V., Shand, B., Gray, E., and et al, Using trust for secure collaboration in uncertain environments. Pervasive Computing, v 2, n 3, Jul.-Sep. 2003, pp. 52-61.

[16] Carter, D., and Katz, A., Computer crime: An emerging challenge for law enforcement, FBI Law Enforcement Bulleting, Dec. 1996, pp. 1-8.

[17] Chalmers, R., and Almeroth, K., A security architecture for mobility-related services. Wireless Personal Communications, v 29, n 3-4, Jun. 2004, pp. 247-61.

[18] Chang, R., Defending against flooding-based, distributed denial-of-service attacks: A tutorial. IEEE Communications Magazine, v 40, n 10, 2002, pp.42–51.

[19] Chen, H., Finin, T., Anupam, J., Kagal, L., Perich, F., and Chakraborty, D., Intelligent agents meet the semantic Web in smart spaces. IEEE Internet Computing, v 8, n 6, Nov.-Dec. 2004, pp. 69-79.

[20] Cicirello, V., Peysakhov, M., Anderson, G., Naik, G., Tsang, K., Regli, W., and Kam, M., Designing dependable agent systems for mobile wireless networks. IEEE Intelligent Systems, v 19, n 5, Sept.-Oct. 2004, pp. 39-45.

[21] CISCO Company. URL: http://www.cisco.com

[22] Coffman, K., and Odlyzko, A., Growth of the internet. Optical Fiber Telecommunications IVB, I. P. Kaminow and T. Li, Eds, San Diego, CA: Academic, 2002, pp. 17–56.

[23] Comer, D., Computer networks and internet. Upper Saddle River, N.J. : Prentice Hall, 1999. ISBN: 0130836176.

[24] Comer, D., Internetworking with TCP/IP: Vol. I -principles, protocols, and architecture. Englewood Cliffs, NJ: Prentice Hall, 1988.

[25] Comer, D., and Stevens, D., Internetworking with TCP/IP Vol. II: design, implementation, and internals. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1994.

[26] CSI/FBI Computer Crime and Security Survey, Computer Security Institute (2005).

[27] Da Silva, A., Loureiro, A., Martins, M., Ruiz, L., Rocha, B., and Wong, H., Decentralized intrusion detection in wireless sensor networks. Proceedings of the First ACM International Workshop on Quality of Service and Security in Wireless and Mobile Networks, Montreal, Canada, 2005, pp. 16-23.

[28] Dasgupta, D., Immunity-based intrusion detection system: A general framework. Proceedings of the 22nd National Information Systems Security Conference, Arlington, USA, 1999, v 1, pp. 147-60.

[29] Date, C., An introduction to database systems. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[30] Day, J., and Zimmerman, H., The OSI reference model. Proceedings of the IEEE, v 71, n 12, Dec. 1983, pp. 1334-40.

[31] Debar, H., Dacier, M., and Wespi, A., A revised taxonomy for intrusion-detection systems. Annales des Telecommunications, v 55, n 7-8, Jul.-Aug. 2000, pp. 361-78.

[32] Demers, A., Research issues in ubiquitous computing. Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, USA, 1994, pp. 2-8.

[33] Demirkol, I., Alagoz, F., Delic, H., and Ersoy, C., Wireless sensor networks for intrusion detection: packet traffic modeling. IEEE Communications Letters, v 10, n 1, Jan. 2006, pp. 22-4.

[34] Denning, P., Electronic Commerce, in Internet Besieged. D. Denning and P. Denning Ed., ACM Press, USA, 1998, pp.377-88.

[35] Dewdney, A., Computer recreations: Of worms, viruses and core war. Scientific American, Mar. 1989, pp. 110.

[36] Diaz, A., Merino, P., Rivas, F., Kulkarni, U., Vadavi, J., Joshi, S., and Yardi, A., Mobile and ubiquitous objects. IEEE Pervasive Computing, v 5, n 3, Jul.-Sep. 2006, pp. 57-9.

[37] Dourish, P., Grinter, R., de la Flor, J., and Joseph, M., Security in the wild: user strategies for managing security as an everyday, practical problem. Personal and Ubiquitous Computing, v 8, n 6, 2004, pp. 391-401.

[38] Dousse, O, Tavoularis, C., and Thiran, P., Delay of intrusion detection in wireless sensor networks. Proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'06), Florence, Italy, 2006, pp. 155-65.

[39] Dritsas, S., Gritzalis, D., and Lambrinoudakis, C., Protecting privacy and anonymity in pervasive computing: trends and perspectives. Telematics and Informatics, v 23, n 3, Aug. 2006, pp. 196-210.

[40] Du, W., Fang, L., and Peng, N., LAD: Localization anomaly detection for wireless sensor networks. Journal of Parallel and Distributed Computing, v 66, n 7, Jul. 2006, pp. 874-86.

[41] Du, Y., Wang, H., and Pang, Y., Design of a distributed intrusion detection system based on independent agents. Proceedings of International Conference on Intelligent Sensing and Information Processing, Chennai, India, 2004, pp. 254-7.

[42] Edwards, W., Discovery systems in ubiquitous computing. IEEE Pervasive Computing, v 5, n 2, Apr./Jun., 2006, pp. 70-77.

[43] English, C., Terzis, S., and Nixon, P., Towards self-protecting ubiquitous systems: monitoring trust-based interactions. Personal and Ubiquitous Computing, v 10, n 1, 2006, pp. 50-4.

[44] Esparza, O., Soriano, M., Munoz, J., and Forne, J., A protocol for detecting malicious hosts based on limiting the execution time of mobile agents. Proceedings of Eighth IEEE International Symposium on Computers and Communication, v 1, Kemer-Antalya, Turkey, Jun.-Jul. 2003, pp. 251-6.

[45] E-Ink homepage – Technology, URL:http://www.eink.com/technology/index.html

[46] Fan, W., Lee, W., Salvatore, J., and Miller, M., A multiple model cost-sensitive approach for intrusion detection. Proceedings of the Eleventh European Conference of Machine Learning, Barcelona, Spain, May 2000, pp. 148-156.

[47] Fano, A., and Gershman, A., The future of business services in the age of ubiquitous computing. Communications of the ACM, v 45, n 12, Dec. 2002, pp. 83-7.

[48] Farkas, K., and et al, Real-time service provisioning for mobile and wireless networks. Computer Communications, v 29, n 5, 6 Mar. 2006, pp. 540-50.

[49] Feng, Z., Mutka, M., and Ni, L., A private, secure, and user-centric information exposure model for service discovery protocols. IEEE Transactions on Mobile Computing, v 5, n 4, Apr. 2006, pp. 418-29.

[50] Fergus, P., Merabti, M., Hanneghan, M., Taleb-Bendiab, A., and Minghwan, A., A semantic framework for self-adaptive networked appliances. Proceedings of IEEE Consumer Communications & Networking Conference (CCNC'05). Las Vegas, USA, 2005, pp. 229 - 34.

[51] Fu, P., Zhang, D., Wang, L., and Duan, Z., Intelligent hierarchical intrusion detection system for secure wireless ad hoc network. Proceedings of the Second International Symposium on Neural Networks, ISNN 2005, LNCS, v 3498, n 3, Chongqing, China, 2005, p 482-7.

[52] Fujitsu e-paper: Changing the way we read, URL:http://www.fujitsu.com/global/about/rd/200509epaper.html

[53] Fukase, M., Akaoka, R., Liu, L., Cheng, T., and Sato, T., Hardware cryptography for ubiquitous computing. International Symposium on Communications and Information Technologies 2005, Tetuan, Morocco, 2005, pp. 478-81.

[54] Gaia: Active spaces for ubiquitous computing. The Gaia homepage, URL:http://choices.cs.uiuc.edu/ActiveSpaces/index.html

[55] Galanxhi, H., and Nah, F., Privacy issues in the era of ubiquitous commerce. Electronic Markets, v 16, n 3, 2006, pp. 222-32.

[56] Gan, F., Joint monitoring of process mean and variance using exponentially weighted moving average control charts. Technometrics, v 37, n 4, 1995, pp. 446-53.

[57] Ghosh, A., Wanken, J., and Charron, F., Detecting anomalous and unknown intrusions against programs. Proceedings of the 1998 Computer Security Applications Conference, IEEE CS Press, Los Alamitos, USA, 1998, pp. 259–67.

[58] GTNetS: Georgia Tech Network Simulator, URL:http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/

[59] Haggerty, J., DiDDeM: A system for early detection of denial-of-service attacks. Ph.D. thesis, Liverpool John Moores University, 2004.

[60] Han, Q., Gutierrez-Nolasco, S., and Venkatasubramanian, N., Reflective middleware for integrating network monitoring with adaptive object messaging. IEEE Network, v 18, n 1, Jan./Feb., 2004, pp. 56-65.

[61] Helmer, G., Wong, J., Honavar, V., Miller, L., and Wang, Y., Lightweight agents for intrusion detection. Journal of Systems and Software, v 67, n 2, Aug. 2003, pp. 109-22.

[62] Hengartner, U., and Steenkiste, P., Access control to people location information. ACM Transactions on Information and Systems Security, v 8, n 4, Nov. 2005, p 424-56.

[63] Hijazi, A., and Nasser, N., Using mobile agents for intrusion detection in wireless ad hoc networks. International Conference on Wireless and Optical Communications Networks, Dubai, UAE, 2005, p 362-6.

[64] Hohl, F., Time limited Blackbox security: Protecting mobile agents from malicious hosts. Mobile Agents and Security, Lecture Notes in Computer Science, v 1419, Springer, Berlin, 1998, pp. 92-113.

[65] Holz, T., Marechal, S., and Raynal, F., New threats and attacks on the World Wide Web. IEEE Security & Privacy, v 4, n 2, Mar.-Apr. 2006, pp. 72-5.

[66] Hong, J., Minimizing security risks in ubicomp systems. Computer, v 38, n 12, Dec. 2005, pp. 118-9.

[67] Hsieh, W., Lo, C., and Chiu, Y., The proactive intrusion prevention for Wireless Local Area Network. International Journal of Mobile Communications, v 4, n 4, 2006, pp. 477-96.

[68] Huang, Y., and Lee, W., A cooperative intrusion detection system for ad-hoc networks. Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks (in association with 10th ACM Conference on Computer and Communications Security), Washington, DC, USA, 2003, pp. 135-47.

[69] Hung, J., Wang, C., Hung, L., Chang, A., and Liao, Y., A smart IDS and response system for the Internet malicious worm. International Journal of Wireless and Mobile Computing, v 1, n 1, 2005, p 70-7.

[70] Hung, L., Giang, P., Zhung, Y., Phuong, T., Lee, S., and Lee, Y., A trust-based security architecture for ubiquitous computing systems. Proceedings of IEEE International Conference on Intelligence and Security Informatics, ISI 2006, LNCS, v 3975, San Diego, USA, 2006, pp. 753-4.

[71] IBM Company. URL: http://www.ibm.com

[72] Iheagwara, C., Blyth, A., and Bennett, M., Architectural and functional issues in systems requirements specifications for wireless intrusion detection systems implementation. Systems Communications 2005, Montreal, Canada, 2005, pp. 434-41.

[73] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F., Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Networking, v 11, n 1, Feb. 2003, pp. 2-16.

[74] Jacoby, G., and Davis, N., Battery-based intrusion detection. Proceedings of IEEE Global Telecommunications Conference, v 4, Dallas, USA, Nov.-Dec. 2004, pp. 2250-5.

[75] Janakiraman, R., Waldvogel, M., and Zhang, Q., Indra: a peer-to-peer approach to network intrusion detection and prevention. Proceedings of the Twelfth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Linz, Austria, Jun. 2003, pp. 226-31.

[76] Jansen, W., Karygiannis, T., Gavrila, S., and Korolev, V., Assigning and enforcing security policies on handheld devices. Proceedings of the Canadian Information Technology Security Symposium, Ottawa, Canada, May 2002.

[77] Jansen, W., Mell, P., Karygiannis, T., and Marks, D., Applying mobile agents to intrusion detection and response. NIST Interim Report (IR) – 6416, 1999.

[78] Java Technology, URL: http://java.sun.com/

[79] Ji, Y., Zhang, P., Hu, Z., Wang, X., Li, Y., and Tang, X., Towards mobile ubiquitous service environment. Wireless Personal Communications, v 38, n 1, Jun. 2006, pp. 67-78.

[80] Jósang, A., Ismail, R., and Boyd, C., A survey of trust and reputation systems for online service provision. Decision Support Systems, v 43, i 2, Mar. 2007, pp. 618-44.

[81] Johnson, D., and Maltz, D., Dynamic source routing in ad-hoc wireless networks. In T. Imielinski and H. Korth, editors, Mobile Computing, Kluwer Academic Publishers, 1996, pp. 153-81.

[82] Jones, A., and Sielken, R., Computer system intrusion detection: A survey. University of Virginia, Computer Science Department, Tech. Rep., 1999.

[83] Kachirski, O., and Guha, R., Intrusion detection using mobile agents in wireless ad hoc networks. Proceedings of IEEE Workshop on Knowledge Media Networking, Kyoto, Japan, 2002, pp. 153-8.

[84] Kasahara, Y., and Yamada, K., NEC's activities for developing business solutions and technology needed for a ubiquitous society. NEC Journal of Advanced Technology, v 1, n 3, Summer 2004, pp. 167-75.

[85] Khoshgoftaar, T., Nath, S., Zhong, S., and Seliya, N., Intrusion detection in wireless networks using clustering techniques with expert analysis. Proceedings of Fourth International Conference on Machine Learning and Applications, Los Angeles, USA, 2005, pp. 120-5.

[86] Kim, S., Tomar, S., Vijaykrishnan, N., Kandemir, M., and Irwin, M., Energy-efficient Java execution using local memory and object co-location. IEE Computers and Digital Techniques, v 151, n 1, 15 Jan. 2004, pp. 33-42.

[87] Krugel, C., and Toth, T., Flexible, mobile agent based intrusion detection for dynamic networks. Euro. Wireless, Italy, Feb. 2002.

[88] Krugel, C., Toth, T., and Kirda, E., Service specific anomaly detection for network intrusion detection. Proceedings of the ACM Symposium on Applied Computing, Madrid, Spain, 2002, pp. 201-8.

[89] Krugel, C., Toth, T., and Kirda, E., Sparta-A mobile agent based intrusion detection system. Advances in Network and Distributed Systems Security. IFIP TC1 WG11.4. First Annual Working Conference on Network Security, Leuven, Belgium, 2002, pp. 187-98.

[90] Lamsal, P., Requirements for modelling trust in ubiquitous computing and ad hoc networks. Research Seminar on Telecommunications Software, autumn 2002. URL: http://www.tml.tkk.fi/Studies/T-110.557/2002/papers/pradip_lamsal.pdf

[91] Landwehr, C., and Goldschlag, D., Security issues in networks with Internet access. Proceedings of the IEEE, v 85, n 12, 1997, pp.2034-51.

[92] Lee, H., Lightweight wireless intrusion detection systems against DDoS attack. Proceedings of International Conference on Computational Science and Its Applications, ICCSA 2006, LNCS, v 3984, Part V, Glasgow, UK, 2006, pp. 294-302.

[93] Lee, W., Cabrera, J., Thomas, A., Balwalli, N., Saluja, S., and Zhang, Y., Performance adaptation in real-time intrusion detection systems. Proceedings of Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, Springer-Verlag, v 2516, 2002, pp. 252-73.

[94] Lee, W., Fan, W., Miller, M., Stolfo, S., and Zadok, E., Toward cost-sensitive modelling for intrusion detection and response. Journal of Computer Security, v 10, n 1-2, 2002, pp. 5-22.

[95] Li, J., Cordes, D., and Zhang, J., Power-aware routing protocols in ad hoc wireless networks. IEEE Wireless Communications, v 12, n 6, Dec. 2005, pp. 69-81.

[96] Liu, Y., Tian, D., and Wei, D., A wireless intrusion detection method based on neural network. Proceedings of the Seventh IASTED International Conference on Advances in Computer Science and Technology, Puerto Vallarta, Mexico, 2006, pp. 207-11.

[97] Liu, Y., Comaniciu, C., and Hong, M., Modelling misbehaviour in ad hoc networks: a game theoretic approach for intrusion detection. International Journal of Security and Networks, v 1, n 3-4, 2006, pp. 243-54.

[98] Long, M., and Wu, C., Energy-efficient and intrusion-resilient authentication for ubiquitous access to factory floor information. IEEE Transactions on Industrial Informatics, v 2, n 1, Feb. 2006, pp. 40-7.

[99] Lunt, T., Tamaru, A., Gilham, F., Jagannathan, R., Neumann, P., Javitz, H., Valdes, A., and Garvey, T., A real-time intrusion detection expert system (IDES) - final technical report. Computer Science Laboratory, SRI International, Menlo Park, California, USA, Feb. 1992.

[100] Manoj, B., Sekhar, A., and Siva Ram Murthy, C., On the use of limited autonomous mobility for dynamic coverage maintenance in sensor networks. Computer Networks, v 51, n 8, Jun, 2007, pp. 2126-43.

[101] Marks, D., Mell, P., and Stinson, M., Optimizing the scalability of network intrusion detection system using mobile agents. Journal of Network and Systems Management, v 12, n 1, Mar. 2004, pp. 95-110.

[102] Mattern, F., The vision and technical foundation of ubiquitous computing. Upgrade, v 2 n 5, Oct. 2001, pp. 2-6.

[103] McQuillan, J., Richer, I., and Rosen, E., The new routing algorithm for the ARPANET. IEEE Transactions on Communications, v COM-28, n 5, May 1980, pp. 711-9.

[104] Microsoft .NET, URL: http://www.microsoft.com

[105] Mishra, A., Nadkarni, K., and Patcha, A., Intrusion detection in wireless ad-hoc networks. IEEE Wireless Communications, v 11, n 1, Feb. 2004, pp. 48-60.

[106] Moore, G., Cramming more components onto integrated circuits, Electronics, v 38, 1965, pp. 114-7.

[107] Moskowitz, I., Kang, M., Chang, L., and Longdon, G., Randomly roving agents for intrusion detection. Proceedings of the IFIP Fifteenth Annual Working Conference on Database and Application Security, Ontario, Canada, 2001, pp. 135-49.

[108] Mukkamala, S., and Sung, A., Artificial intelligent techniques for intrusion detection. Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC'03). Conference Theme - System Security and Assurance, v 2, Washington, DC, USA, 2003, pp. 1266-71.

[109] Navidi, W., and Camp, T., Stationary distributions for the random waypoint mobility model. IEEE Transactions on Mobile Computing, v 3 n 1, Jan.-Mar. 2004, pp. 99-108.

[110] Ohkubo, M., Suzuki, K., and Kinoshita, S., RFID privacy issues and technical challenges. Communications of the ACM, v 48, n 9, Sept. 2005, pp. 66-71.

[111] Ould-Ahmed-Vall, E., Riley, G., Heck, B., and Reddy, D., Simulation of large-scale sensor networks using GTSNetS. Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05), Atlanta, USA, 2005, pp. 211-8.

[112] Peddireddy, T., and Vidal, J., Multiagent network security system using FIPA-OS. Proceedings of IEEE SoutheastCon, South Carolina, USA, 2002, pp. 229-33.

[113] Petrovic, S., Vulnerabilities in wireless networks and intrusion detection. Telektronikk, v 101, n 1, 2005, pp. 86-91.

[114] Pitts, J., and Schormans, J., Introduction to ATM design and performance: With applications analysis software. John Wiley & Sons, Inc. New York, NY, USA, 1996.

[115] Porras, P., and Neumann, P., EMERALD: Event monitoring enabling responses to anomalous live disturbances. Proceedings of 20th NIST-NCSC National Information Systems Security Conference, National Institute of Standards and Technology, 1997, pp.353-65.

[116] Postel, J., Simple Mail Transfer Protocol (SMTP). STD 10, RFC 821, USC/ISI, Aug. 1982. URL: http://www.ietf.org/rfc/rfc0821.txt

[117] Postel, J., and Reynolds, J., File Transfer Protocol (FTP). STD 9, RFC 959, USC/ISI, Oct. 1985. URL: http://www.ietf.org/rfc/rfc0959.txt

[118] Power, R., and Forte, D., Wireless, PDA and instant messaging: Achilles' heel? Computer Fraud & Security, v 2005, n 10, Oct. 2005, p 7-9.

[119] Ptacek, T., and Newsham, T., Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Jan. 1998.

[120] Qian, L., Song, B., and Li, X., Detection of wormhole attacks in multi-path routed wireless ad hoc networks: A statistical analysis approach. Journal of Network and Computer Applications, v 30, n 1, Jan., 2007, pp. 308-30.

[121] Qin, X., Lee, W., Lewis, L., and Cabrera, J., Integrating intrusion detection and network management. Proceedings of IEEE/IFIP Network Operations and Management Symposium. Management Solutions for the New Communications World, Florence, Italy, 2002, pp. 329-44.

[122] Ragsdale, D., Carver, C., Humphries, J., and Pooch, U., Adaptation techniques for intrusion detection and intrusion response systems. Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Nashville, USA, Oct. 2000, pp. 2344-9.

[123] Riley, G., The Georgia Tech network simulator. Proceedings of Workshop on Models, Methods, and Tools for Reproducible Network Research (MoMeTools), Karlsruhe, Germany, Aug. 2003, pp.5-12.

[124] Roman, R., Zhou, J., and Lopez, J., Applying intrusion detection systems to wireless sensor networks. 3rd IEEE Consumer Communications and Networking Conference, CCNC 2006, v 1, Las Vegas, USA, 2006, pp. 640-4.

[125] Roussos, G., and Moussouri, T., Consumer perceptions of privacy, security and trust in ubiquitous commerce. Personal and Ubiquitous Computing, v 8, n 6, 2004, pp. 416-29.

[126] Royer, E., and Toh, C.-K., A review of current routing protocols for ad-hoc mobile wireless networks. IEEE Pers. Commun., Apr. 1999, pp. 46–55.

[127] Ryan, T., Statistical methods for quality improvement (2nd Ed, Wiley, 2000). ISBN: 0471197750.

[128] Sagara, K., Nishiki, K., and Koizumi, M., A distributed authentication platform architecture for peer-to-peer applications. IEICE Transactions on Communications, v E88-B, n 3, Mar. 2005, pp. 865-72.

[129] Salus, P., Casting the net: From ARPANET to INTERNET and beyond. Reading, MA: Addison-Wesley, 1995. ISBN: 0201876744.

[130] Sande, T., and Tschudin, C., Protecting mobile agents against malicious hosts. Mobile agents and security. Lecture Notes in Computer Science, v 1419, Springer-Verlag, 1998, pp. 44-60.

[131] Sang, K., and Choon, S., Security threats and their countermeasures of mobile portable computing devices in ubiquitous computing environments. Proceedings of International Conference on Computational Science and Its

Applications, ICCSA 2005, LNCS, v 3483, Part IV, Singapore, 2005, pp. 79-85.

[132] Schmidt, M., and Arnett, K., Spyware: a little knowledge is a wonderful thing. Communications of the ACM, v 48, n 8, Aug. 2005, pp. 67-70.

[133] Schmoyer, T., Yu, X., and Owen, H., Wireless Intrusion detection and response - A case study using the classic man-in-the-middle attack. Proceedings of IEEE Wireless Communications and Networking Conference, v 2, Atlanta, USA, Mar. 2004, pp. 883-8.

[134] Shand, B., Dimmock, N., and Bacon, J., Trust for ubiquitous, transparent collaboration. Wireless Networks, v 10, n 6, Nov. 2004, pp. 711-21.

[135] Shin, M., Ma, J., Mishra, A., and Arbaugh, W., Wireless network security and interworking. Proceedings of the IEEE, v 94, n 2, Feb. 2006, pp. 455-66.

[136] Singh, S., Woo, M., and Raghavendra, C., Power-aware routing in mobile ad hoc networks. Proceedings of ACM MobiCom'98, Dallas, USA, 1998, pp. 181-90.

[137] Sluzek, A., Annamalai, P., and Islam, M., A wireless sensor network for visual detection and classification of intrusions. WSEAS Transactions on Circuits and Systems, v 4, n 12, Dec. 2005, pp. 1855-60.

[138] Snapp, S., Brentano, J., Dias, G., Goan, T., and et al, A system for distributed intrusion detection. COMPCON Spring 1991. Digest of Papers, San Francisco, USA, 1991, pp. 170-6.

[139] Sobh, T., Wired and wireless intrusion detection system: Classifications, good characteristics and state-of-the-art. Computer Standards & Interfaces, v 28, n 6, Sept. 2006, pp. 670-94.

[140] Spafford, E., and Zamboni, D., Intrusion detection using autonomous agents. Computer Networks, v 34, n 4, Oct. 2000, pp. 547-70.

[141] Srinivasan, T., Seshadri, J., Jonathan, J., and Chandrasekhar, A., A system for power-aware agent-based intrusion detection (SPAID) in wireless ad hoc networks. 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, LNCS, v 3654, Storrs, USA, 2005, pp. 153-62.

[142] Stajano, F., Security for ubiquitous computing. Wiley, 2002. ISBN 0470844930.

[143] Stallings, W., Cryptography and network security: principles and practice. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1998.

[144] Stallings, W., Pretty Good Privacy. ConneXions, v 8, n 12, Dec. 1994, pp. 2-11.

[145] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., and Zerkle, D., GrIDS-A graph based intrusion detection system for large networks. Proceedings of the 19th National Information Systems Security Conference, v 1, National Institute of Standards and Technology, Oct. 1996, pp.361-70.

[146] Stevens, W., TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley, 1994.

[147] Strassner, M., and Schoch, T., Today's impact of ubiquitous computing on business processes. Proceedings of the First International Conference on Pervasive Computing, Zurich, Switzerland, 2002.

[148] Takizawa, O., Ubiquitous communications technology for disaster mitigation. Journal of the National Institute of Information and Communications Technology, v 52, n 1-2, Mar.-Jun. 2005, pp. 235-58.

[149] Tanenbaum, A., Computer networks, 4th Ed. Pearson Education International, Upper Saddle River, NJ, USA, 2002.

[150] Thompson, H., Whittaker, J., and Andrews, M., Intrusion detection: Perspectives on the insider threat. Computer Fraud & Security, Jan. 2004, pp. 13-5.

[151] Toh, C.-K., Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks. IEEE Communications Magazine, v 39, n 6, Jun. 2001, pp. 138-47.

[152] Tseng, H., and Culpepper, B., Sinkhole intrusion in mobile ad hoc networks: The problem and some detection indicators. Computers and Security, v 24, n 7, October, 2005, pp. 561-70.

[153] USA Department of Defense. Trusted computer system evaluation criteria. National Computer Security Center, DOD 5200.28-STD, Dec. 1985.

[154] Vogt, H., Small worlds and the security of ubiquitous computing. Proceedings of Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, WoWMoM 2005, Taormina, Greece, 2005, pp. 593-7.

[155] Weiser, M., The computer for the 21st century. Scientific American (International Edition), v 265, n 3, Sept. 1991, pp. 66-75.

[156] Wilder, A., and Shanmugasundaram, V., An introduction to intrusion detection in the wireless environment. Proceedings of the 20th International Conference Computers and their Applications, New Orleans, USA, 2005, pp. 109-14.

[157] Yamada, S., and Kamioka, E., Access control for security and privacy in ubiquitous computing environments. IEICE Transactions on Communications, v E88-B, n 3, Mar. 2005, pp. 846-56.

[158] Ye, N., and Chen, Q., An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. Quality and Reliability Eng. Int'l, v 17, n 2, 2001, pp. 105-12.

[159] Ye, N., Chen, Q., Emran, S., and Noh, K., Chi-square statistical profiling for anomaly detection. IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, USA, Jun. 2000.

[160] Zhang, Y., and Lee, W., Intrusion detection in wireless ad-hoc networks. Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, Boston, USA, 2000, pp. 275-83.

[161] Zhang, Y., Lee, W., and Huang, Y., Intrusion detection techniques for mobile wireless networks. Wireless Networks, v 9, n 5, Sept. 2003, pp. 545-56.

[162] Zhang, Z., Manikopoulos, C., and Jorgenson, J., Architecture of generalized network service anomaly and fault thresholds. Proceedings of 4th IFIP/IEEE International Conference on Management of Multimedia Networks and Services, (MNS 2001), Chicago, USA, Oct.- Nov., 2001, pp. 241-55.

[163] Zhong, S., Khoshgoftaar, T., and Nath, S., A clustering approach to wireless network intrusion detection. Proceedings of 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05), Hong Kong, China, 2005, pp. 190-6.

[164] Zhong, S., Song, Q., Cheng, X., and Zhang, Y., A safe mobile agent system for distributed intrusion detection. Proceedings of International Conference on Machine Learning and Cybernetics, v 4, Xi'an, China, 2003, pp. 2009-14.

[165] Zhou, S., Qin, Z., Luo, X., Zhang, X., Zhang, F., and Liu, J., Cost-based intelligent intrusion detection and response: design and implement. Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, Chengdu, China, 2003, pp. 166-70.

[166] Zincir, I., Furnell, S., and Phippen, A., Intrusion detection via behavioral profiling on mobile and wireless networked devices. Eleventh Annual Scientific Conference on Web Technology, New Media Communications and Telematics Theory Methods, Tools and Applications (EUROMEDIA'05), Toulouse, France, 2005, pp. 67-71.

APPENDIX A

SIMULATION CODE

This Appendix provides the code used for network simulation. The simulated environment is described in chapter nine. It is written in C++ programming language under GTSNetS.

smartspace_normal.cc

```
/* Written by Bo Zhou, Liverpool JMU, 2006
   Simulation of a smart space with one head node, ten user nodes and forty
   service nodes. Energy consumptions are recorded.
*/

#include "simulator.h"
#include "node.h"
#include "wlan.h"
#include "ratetimeparse.h"
#include "application-cbr-sn.h"
#include "udp.h"
#include "routing-dsr.h"
#include "routing-nvr.h"
#include "routing-aodv.h"
#include "wireless-grid-rectangular.h"
#include "trace.h"
#include "application-sntest.h"
#include "mobility-random-waypoint.h"
#include "servicenode-CBR.h"
#include <sstream>
#include <fstream>
#include "trace-sn.h"
#include "battery.h"
#include "sensors.h"
```

```
#include "sensor.h"
#include "node-sn.h"
#include "interface-wireless.h"
#include "ratetimeparse.h"

#define ANIMATION_ON
#define XWIDTH              150
#define YWIDTH              150
#define NO_NODE             51
#define NO_HNODE            1
#define NO_UNODE            10
#define NO_SNODE            40
#define NO_SNODECBR         20
#define NO_SNODECBR_P       20
#define RADIO_RANGE         30
#define SIM_TIME            20000
#define UNODETIME           3000
#define INIT_ENERGY         2000   //Initial energy of 2 Joules (2000 mJ)
#define ROUTING_DNVR        1
#define ROUTING_DSR         2
#define ROUTING_AODV        3

using namespace std;

// Simple timer class for random choose scenario
class RandomChoose : public Timer {
public:
  virtual void Timeout(TimerEvent*);   // Called when timer expires
  RandomChoose(Node**, ServiceNodeCBR*, ServiceNodeCBR*, Node*,
                                  double*, double*, Time_t*, Time_t*, Energy_t*);
  int ActiveNode();
  void ProcessRecord(string, Node*, Node*, PortId_t);
  Node* getMidNode(Node*);
  PortId_t getMidPort(Node*);
  string status;
  Node** AllNode;
  double* si;     //energy consumption rate at idle status
  double* sb;    //energy consumption rate at busy status
```

```
    Time_t* st;   //safetime since last known abnormal activities
    Time_t* pt;   //previous time since last status changed
    Energy_t* pe;   //previous energy left since last status changed

private:
        ServiceNodeCBR* CBRNode;
        ServiceNodeCBR* CBRNode_P;
        Node* UserNode;
        int rNumber;
        Node* midNode;
        PortId_t midPort;
        Uniform urvNodeNumber, urvBusyTime, urvIdleTime, attackTime, CBRRate;
        Uniform urvCBRP[NO_SNODECBR_P];
                    //parameter range of CBR nodes
        bool nwkdied, unodeEnergyRecorded;
        ofstream eventRecords;   //recorder for entire system
};

RandomChoose::RandomChoose(Node** n, ServiceNodeCBR* cbrnode,
                    ServiceNodeCBR* cbrnode_p, Node* usernode, double* s1,
                        double* s2, Time_t* t1, Time_t* t2, Energy_t* e)
{
        urvNodeNumber=Uniform(0, NO_SNODE);
    urvBusyTime=Uniform(3, 10);   //How long the user will use the service node.
        attackTime=Uniform(20,30);     //During of launched attack
    urvIdleTime=Uniform(5, 10); //idle time between services. similar to busytime
        CBRRate=Uniform(1,10);      //CBR rate is randomly generated too.
        urvCBRP[0]=Uniform(18,25);
        urvCBRP[1]=Uniform(30,50);
        urvCBRP[2]=Uniform(65,95);
        AllNode=n;
        UserNode=usernode;
        CBRNode=cbrnode;
        CBRNode_P=cbrnode_p;
        UserNode->SetTrace(Trace::DEFAULT);
        sb=s1;
        si=s2;
        st=t1;
```

```
pt=t2;
pe=e;
nwkdied=false; //network is not died yet
unodeEnergyRecorded=false; //user node's energy consumption rate is
                                    //calculated during the process, only once.
//open events file
eventRecords.open( "events.txt", ios::app);
if (!eventRecords){
        cerr<<"File could not be opened"<<endl;
            exit(1);
        }
 }

int RandomChoose::ActiveNode(){
        int nNum;
        int m=0;
        while (m < 2*NO_SNODE){
           nNum=(int)floor(urvNodeNumber.Value());//make it 0~5
           if(!static_cast<NodeSN*>(AllNode
                        [nNum+1+NO_HNODE+NO_UNODE])->IsDead()) break;
           else m++;
         }
         if (m==2*NO_SNODE) nNum=NO_SNODE+1;
             return nNum;
  }

Node* RandomChoose::getMidNode(Node* n){
    Node* node=n;
    Meters_t dist;
    int nodeNum=0;
    double B=0;//energy
    double T=0;//trust level
    double R=0;//busy/idle ratio
    double tempMetric=0;
    double maxMetric=0;
    Energy_t initEnergy=INIT_ENERGY;//currently we use the same configure
    Energy_t curEnergy;
    Time_t now = Simulator::Now();
```

```
for (int i = 1; i <= NO_NODE; i++) {
   dist=node->Distance(AllNode[i]);
   if ((dist<RADIO_RANGE))
   {
       curEnergy=static_cast<NodeSN*>
                (AllNode[i])->getSNBattery()->GetRemainingEnergy();
       if (i<=(NO_HNODE+NO_UNODE)) initEnergy=INIT_ENERGY*2;
                 //head node and user node have double battery capacity
       else initEnergy=INIT_ENERGY;
       cout<<"node ID:"<<AllNode[i]->Id()<<" "<<i<<"remainEnergy:"
            <<curEnergy<<" pt:"<<pt[i-1]<<" pe:"<<pe[i-1]<<" now:"
            <<now<<endl;
       double ecrate=(initEnergy-curEnergy)/now;
       B=curEnergy/ecrate;   // currently do not need to consider d^2 and t
                             //since the radio rang didn't really change
       B=B/1000;   //to confine the range of B to double figures
       //note: prevent choosing node which has remaining energy less than 0
        //but bigger tempMetric(caused by square).
       R=(ecrate-si[i-1])/(sb[i-1]-ecrate);
   if ((ecrate<=si[i-1])||(R<0.1)) R=0.1;   //avoid too small and minus value
   else if ((ecrate>=sb[i-1])||(R>10)) R=10; //avoid too big and minus value
       double x=ceil((now-st[i-1])/1000);
       if (x>10) x=10;
       double y=((pe[i-1]-curEnergy)/(now-pt[i-1]))/si[i-1];
       if (y<1) y=1;
       T=x/y;
       tempMetric=(B*T)/R;
       cout<<" ecrate:"<<ecrate<<" idle:"<<si[i-1]<<" busy:"<<sb[i-1]
            <<" x:"<<x<<" y:"<<y<<endl;
       cout <<" B:"<<B<<" R:"<<R<<" T:"<<T
            <<" tempMetric:"<<tempMetric<<endl;
       if ((curEnergy>1)&&(tempMetric>maxMetric)){
                             //1 used to ensure the node is not died
       maxMetric=tempMetric;
       nodeNum=i;
   }
   //energy cost on distribution of the hybrid metric
   e = (static_cast<NodeSN*>(AllNode[i])->getPerBitEnergy())*5;
```

```
        static_cast<NodeSN*>(AllNode[i])->updateComputingEnergy(e);
    }
}
cout<<"chosen node ID:"<<AllNode[nodeNum]->Id()
      <<" number i:"<<nodeNum<<endl;
if (nodeNum==0){ //all neighbour nodes are died,e.g.network died
    if(nwkdied==false){
        Time_t now=Simulator::Now();
        ofstream deathRecords;
        deathRecords.open( "death.txt", ios::app);//open mode is append.
        if (!deathRecords){
            ceer<<"File could not be opened"<<endl;
            exit(1);
        }
        deathRecords<<now<<" network is died"<<endl;
        nwkdied=true; //network is already died
    }
    return node;
}
  else return AllNode[nodeNum];
}
PortId_t RandomChoose::getMidPort(Node* n){
   Node* node=n;
   PortId_t p=1000+node->Id();
   return p;
}
void RandomChoose::ProcessRecord(string s, Node* sn, Node* mn, PortId_t mp){
   string status=s;
   Node* snode=sn;
   Node* mnode=mn;
   //mnode=AllNode[1];//use head node as proxy node
   //mnode=snode;//use service node itself as proxy node
   PortId_t mport=mp;
   mport=1000+mnode->Id();
   Size_t ssize=status.length();
   IPAddr_t mdst=mnode->GetIPAddr();
   Energy_t e;
   cout<<status<<" size of s is:"<<ssize<<endl;
```

```
cout<<" middle node is:"<<mnode->Id();
cout<<" midle port is:"<<mport<<endl;
L4Protocol* l4Proto=static_cast<NodeSN*>(snode)->
                                        GetApplicationSN()->GetL4();
l4Proto->SendTo(ssize*2, mnode->GetIPAddr(), mport);
if (!static_cast<NodeSN*>(snode)->IsDead()) {
   e = (static_cast<NodeSN*>(snode)->getPerBitEnergy())*ssize;
   e=e*500; //process ids event needs extra processing power.
   static_cast<NodeSN*>(snode)->updateComputingEnergy(e);
   }
   if (!static_cast<NodeSN*>(mnode)->IsDead()) {
      e = 5; //process ids event needs extra processing power.
      static_cast<NodeSN*>(mnode)->updateComputingEnergy(e);
   }
}

void RandomChoose::Timeout(TimerEvent* ev){
   Time_t now = Simulator::Now();
   if ((now>UNODETIME)&&(!unodeEnergyRecorded)){
       unodeEnergyRecorded=true;
      eventRecords<<"======================================"<<endl;
      for (int i = 0; i < NO_UNODE; i++) {
          eventRecords<<"node "<<AllNode[2+i]->Id()<<" "
                        << static_cast<NodeSN*> (AllNode[2+i])
                        ->getSNBattery()->GetRemainingEnergy()<<endl;
       }
}
double busyTime;
busyTime=urvBusyTime.Value();
double idleTime=urvIdleTime.Value();
if (now<(SIM_TIME-100)){
rNumber=ActiveNode();
cout << "Progress " << now << endl;
cout<<"Randomly selected node is: node "
          <<1+rNumber+NO_HNODE+NO_UNODE<<endl;
cout<<"Random busy period is:"<<busyTime<<" seconds"<<endl;
cout<<"Random idle period is:"<<idleTime<<" seconds"<<endl;
if (rNumber<NO_SNODECBR){
```

```
    CBRNode[rNumber].GetApp()->SetRemoteNode(UserNode);
int rateIndex=(int)CBRRate.Value();
CBRNode[rNumber].GetApp()->cbrRate=
                                (Rate_t)Rate("4.096Kb") * rateIndex/5.0;
status=CBRNode[rNumber].SetStatus("on", now);
    midPort=getMidPort(midNode);
ProcessRecord(status, CBRNode[rNumber].GetNode(), midNode,
                                                        midPort);
CBRNode[rNumber].GetApp()->StartApp();
CBRNode[rNumber].GetApp()->Stop(busyTime);
    status=CBRNode[rNumber].SetStatus("off", now+busyTime);
ProcessRecord(status, CBRNode[rNumber].GetNode(), midNode,
                                                      midPort);
pt[rNumber+NO_HNODE+NO_UNODE]=now+busyTime;
     pe[rNumber+NO_HNODE+NO_UNODE]=static_cast<NodeSN*>
            (AllNode[1+rNumber+NO_HNODE+NO_UNODE])->
            getSNBattery()->GetRemainingEnergy();
}
 else if (rNumber<NO_SNODECBR+NO_SNODECBR_P){
     rNumber=rNumber-NO_SNODECBR;
double ranPara=urvCBRP[(rNumber%3)].Value();
CBRNode_P[rNumber].GetApp()->SetRemoteNode(UserNode);
int rateIndex=(int)CBRRate.Value();
CBRNode_P[rNumber].GetApp()->cbrRate=
                            (Rate_t)Rate("4.096Kb") * rateIndex/5.0;
    status=CBRNode_P[rNumber].SetStatus("on", now);
     midNode=getMidNode(CBRNode_P[rNumber].GetNode());
midPort=getMidPort(midNode);
    ProcessRecord(status, CBRNode_P[rNumber].GetNode(), midNode,
          midPort);
     status=CBRNode_P[rNumber].SetStatus("set", now, ranPara);
    ProcessRecord(status, CBRNode_P[rNumber].GetNode(), midNode,
          midPort);
     CBRNode_P[rNumber].GetApp()->StartApp();
CBRNode_P[rNumber].GetApp()->Stop(busyTime);
status=CBRNode_P[rNumber].SetStatus("off", now+busyTime);
ProcessRecord(status, CBRNode_P[rNumber].GetNode(), midNode,
                    midPort);
```

```
            pt[rNumber+NO_HNODE+NO_UNODE+NO_SNODECBR]=
                                    now+busyTime;
            pe[rNumber+NO_HNODE+NO_UNODE+NO_SNODECBR]=
                  static_cast<NodeSN*>(AllNode[1+rNumber+NO_HNODE
+NO_UNODE+NO_SNODECBR])->getSNBattery()->GetRemainingEnergy();
          }else{//all nodes are died
              Time_t now=Simulator::Now();
                ofstream deathRecords;
              deathRecords.open( "death.txt", ios::app);
                    //open mode is append.
              if (!deathRecords){
              cerr<<"File could not be opened"<<endl;
              exit(1);
            }
             deathRecords<<now<<" all nodes are died"<<endl;
           idleTime=SIM_TIME-now-100;
                 //make it ends now. no more random choose procedure
    }//end of else
    Schedule(ev, busyTime+idleTime);
} else {// start record all actions to trace file.
        for (int i = 0; i < NO_SNODECBR; i++) {
              eventRecords<<CBRNode[i].GetRecords();
        }
        for (int i = 0; i < NO_SNODECBR_P; i++) {
              eventRecords<<CBRNode_P[i].GetRecords();
        }
        eventRecords.close();
    }
}

class RecordDeath : public Timer {
public:
   virtual void Timeout(TimerEvent*);   // Called when timer expires
   RecordDeath(Node**);
private:
      Node** AllNode;
      bool ndied[NO_NODE+1];
      Time_t deathTime[NO_NODE+1];
```

```
        Time_t totalTime;
        Time_t meanTime;
        ofstream deathRecords;//recorder for entire system
        int diedNum;
};
RecordDeath::RecordDeath(Node** allnode){
        AllNode=allnode;
        //open events file
        deathRecords.open( "death.txt", ios::app);//open mode is append.
        if (!deathRecords){
                cerr<<"File could not be opened"<<endl;
                exit(1);
        }
        deathRecords<<"====================================="<<endl;
        for (int i = 0; i <= NO_NODE; i++)  ndied[i]=false;
        diedNum=0;
        totalTime=0;
        meanTime=0;
}
void RecordDeath::Timeout(TimerEvent* ev) {
Time_t now = Simulator::Now();
if (now<(SIM_TIME-50)) {
  for (int i = 0; i < NO_NODE; i++){
    if ((!ndied[1+i]) && (static_cast<NodeSN*> (AllNode[1+i])->IsDead())){
      deathRecords<<now<<" node "<<AllNode[1+i]->Id()<<" is died"<<endl;
      ndied[1+i]=true;
      deathTime[1+i]=now;
      diedNum++;
    }
 }
 if(diedNum<(NO_NODE/2))   Schedule(ev, 1.5);
 else {
deathRecords<<now<<" The network(half Nodes) is died "<<endl;
for (int i = 0; i < NO_NODE; i++){
  if(ndied[1+i]) totalTime=totalTime+deathTime[1+i];
}
deathRecords<<"Total life time is: "<<totalTime<<endl;
deathRecords<<"Died node number is:"<<diedNum<<endl;
```

```
        meanTime=totalTime/diedNum;
        deathRecords<<"Everage life time is: "<<meanTime<<endl;
        //standard=maybe calculate standard deviation later
        Schedule(ev,SIM_TIME-now-40);
      }
    }else deathRecords.close();
}

int main(int argc, char** argv)
{
    /* parameters setting
       1. ad hoc routing protocol type
       2. seed
    */
    int routingProto = ROUTING_DSR;
    Seed_t seed = 1;
    if (argc > 1) {
        if (!strcmp("dnvr", argv[1])) routingProto = ROUTING_DNVR;
        else if (!strcmp("aodv", argv[1])) routingProto = ROUTING_AODV;
    }
    if (argc > 2) seed = atoi(argv[2]);
    Random::GlobalSeed(seed, seed, seed, seed, seed, seed);

    Simulator s;
    s.HasMobility(true);

    // trace file
    Trace* gs = Trace::Instance();
    gs->Open("smartspace_normal.txt");

    TraceSN* ts = TraceSN::Instance();
  ts->Open("smartspace_normal_sn.txt");
  ts->OnSN();
  gs->IPDotted(true);

  //TCP::LogFlagsText(true);
   // Log flags in text mode, e.g. show 'SYN' insteadof '0x02'
  //TCP::TCPHeader->DetailOff(TCP::FID);
```

```
// Increase detail of L3 trace messages
IPV4::Instance()->DetailOn(IPV4::TOTALLENGTH);
                                        //packet total length,inc.data
IPV4::Instance()->DetailOff(IPV4::TTL);//disable time to live column
IPV4::Instance()->DetailOff(IPV4::VERSION);//disable version column IP4 or 6
IPV4::Instance()->DetailOff(IPV4::UID);//packet unique ID. default ON
IPV4::Instance()->DetailOff(IPV4::PROTOCOL);//disable protocol number
//IPV4::Instance()->DetailOn(IPV4::FRAGMENTOFFSET); Default off
IPV4::Instance()->SetTrace(Trace::DEFAULT);

// routing protocol
if (routingProto == ROUTING_DSR) {
   Routing::SetRouting(new RoutingDSR);   // set DSR to default routing
}
else if (routingProto == ROUTING_DNVR) {
   Routing::SetRouting(new RoutingNVR);   // set NVR to default routing
}

Energy_t initEnergy =INIT_ENERGY;
Time_t recInterval = 500;
WirelessLink wlink((NO_NODE), IPAddr("192.168.0.0"), MASK_ALL,
                                recInterval, initEnergy, MODEL1);
Uniform urvx(0, (XWIDTH-RADIO_RANGE)),
                                        urvy(0, (YWIDTH-RADIO_RANGE));
//Create Grid to allow random waypoint mobility,
//+-1 to make sure the mobile node virtually will not move out of grid
Uniform borderx(0, XWIDTH-RADIO_RANGE-1),
                                        bordery(0, YWIDTH-RADIO_RANGE-1);
WirelessGridRectangular g(Location(1,1), Constant(NO_NODE), borderx,
                                        bordery, IPADDR_NONE, false);
Node* n[NO_NODE+1];
PortId_t rPort[NO_NODE+1];
PortId_t lPort[NO_NODE+1];
InterfaceWirelessSN* iface[NO_NODE+1];
for (int i = 0; i <=NO_NODE; i++) {
   n[i] = wlink.GetNode(i);
   n[i]->SetRadioRange(RADIO_RANGE);
   n[i]->SetLocation(urvx.Value(), urvy.Value());
```

```
rPort[i]=1000+i;
lPort[i]=1000+i;
if (i==0){
   //initialize node n0. It is a sink node automatically generated by wlink.
   n[i]->SetRadioRange(0);
   n[i]->SetLocation(XWIDTH, YWIDTH);
   n[i]->SetTrace(Trace::DISABLED);
   static_cast<NodeSink*> (n[i])->TraceSNOff(NodeSink::SINKNODE_ID);
   static_cast<NodeSink*> (n[i])->
                  TraceSNOff(NodeSink::NODE_SENSING_ENERGY);
static_cast<NodeSink*> (n[i])->
                  TraceSNOff(NodeSink::NODE_RELAYING_ENERGY);
static_cast<NodeSink*> (n[i])->
                  TraceSNOff(NodeSink::NODE_COMPUTING_ENERGY);
static_cast<NodeSink*> (n[i])->
   TraceSNOff(NodeSink::NODE_RELAY_OVERHEAD_ENERGY);
static_cast<NodeSink*> (n[i])->
    TraceSNOff(NodeSink::NODE_COMP_OVERHEAD_ENERGY);
static_cast<NodeSink*> (n[i])->
                  TraceSNOff(NodeSink::NETW_TOTAL_ENERGY);
static_cast<NodeSink*> (n[i])->
                  TraceSNOff(NodeSink::NETW_SENSING_ENERGY);
static_cast<NodeSink*> (n[i])->
                  TraceSNOff(NodeSink::NETW_RELAYING_ENERGY);
static_cast<NodeSink*> (n[i])->
                 TraceSNOff(NodeSink::NETW_COMPUTING_ENERGY);
static_cast<NodeSink*> (n[i])->
    TraceSNOff(NodeSink::NETW_RELAY_OVERHEAD_ENERGY);
static_cast<NodeSink*> (n[i])->
    TraceSNOff(NodeSink::NETW_COMP_OVERHEAD_ENERGY);
static_cast<NodeSink*> (n[i])->
                 TraceSNOff(NodeSink::NETW_LIFETIME);
 static_cast<NodeSink*> (n[i])->TraceSNOff(NodeSink::SENSED_DATA);
 static_cast<NodeSink*> (n[i])->TraceSNOff(NodeSink::NODE_X);
 static_cast<NodeSink*> (n[i])->TraceSNOff(NodeSink::NODE_Y);
 static_cast<NodeSink*> (n[i])->TraceSNOn(NodeSink::TIME);
 static_cast<NodeSink*> (n[i])->TraceSNOn(NodeSink::SENSNODE_ID);
}
```

```
else{
   //static_cast<NodeSN*> (n[i])->SetApplicationSN (app[i]);
   static_cast<NodeSN*> (n[i])->getSNBattery()->SetMinEnergy(1);
   static_cast<NodeSN*> (n[i])->setSize(32);
   static_cast<NodeSN*> (n[i])->setPerBitEnergy(0.0002);
      //parameter to define how much energy are used for each packet
      //processed. kind of at application layer. The physical layer cost is
       //defined by interface initialization.
   static_cast<NodeSN*> (n[i])->setComputePower(0.00001);
   iface[i] = static_cast<InterfaceWirelessSN*>(static_cast<NodeSN*>
                                        (n[i])->GetSNIface());
   iface[i]->setTxPower(0.000001);
   iface[i]->energyModel = MODEL1;
   iface[i]->SetPerBitTxEnergy(0.000045);
   iface[i]->SetPerBitM2TxEnergy(0.00001);
                  //in micro Joules : 10 pJ/bit/m2 for attenuation factor of 2
   iface[i]->SetAttenuationFactor(2);
   iface[i]->SetPerBitRxEnergy(0.000135);

}
  }

CBRApplicationSN* app[NO_HNODE+NO_UNODE+1];

//Initiate array of Head Nodes
for (int i = 0; i < NO_HNODE; i++) {
    n[1+i]->Color(Qt::blue);
    app[1+i]=new CBRApplicationSN(n[0], rPort[1+i], lPort[1+i], 0, 512, UDP());
    static_cast<NodeSN*> (n[1+i])->SetApplicationSN (app[1+i]);
    static_cast<NodeSN*>(n[1+i])->getSNBattery()->
                    SetRemainingEnergy(INIT_ENERGY*2);
}
 //Initiate array of User Nodes
  for (int i = 0; i < NO_UNODE; i++) {
    n[1+NO_HNODE+i]->Color(Qt::red);
    n[1+NO_HNODE+i]->Shape(Node::CIRCLE);
    n[1+NO_HNODE+i]->AddMobility(RandomWaypoint(g,      Uniform(1200,1500),
Uniform(0,1)));//first is the thinking time, second is the velocity
```

```
app[1+NO_HNODE+i]=new CBRApplicationSN(n[0], rPort[1+NO_HNODE+i],
                        lPort[1+NO_HNODE+i], 0, 512, UDP());
    static_cast<NodeSN*> (n[1+NO_HNODE+i])->
                        SetApplicationSN (app[1+NO_HNODE+i]);
    static_cast<NodeSN*>(n[1+NO_HNODE+i])->getSNBattery()->
                        SetRemainingEnergy(INIT_ENERGY*2);
  }

  //Initiate array of CBR service node without parameter only on off
  Rate_t cbrrate[] = {(Rate_t)Rate("4.096Kb") * 0,
                        (Rate_t)Rate("4.096Kb") * 0, (Rate_t)Rate("4.096Kb") * 0};
  Size_t cbrpsize[]={512, 512, 512};
  ServiceNodeCBR cbrnode[NO_SNODECBR];
  for (int i = 0; i < NO_SNODECBR; i++) {
    cbrnode[i]=ServiceNodeCBR(n[0],
        wlink.GetNode(1+NO_HNODE+NO_UNODE+i),
        rPort[1+NO_HNODE+NO_UNODE+i],
     lPort[1+NO_HNODE+NO_UNODE+i], cbrrate[(i%3)], cbrpsize[(i%3)]);
  }

  //Initiate array of CBR service node with one parameter on off and set
  Rate_t cbrrate_p[] = {(Rate_t)Rate("4.096Kb") * 0,
                        (Rate_t)Rate("4.096Kb") * 0, (Rate_t)Rate("4.096Kb") * 0};
  Size_t cbrpsize_p[]={512,512,512};
  double cbr_para[]={20, 40, 80}; //default parameter value
  ServiceNodeCBR cbrnode_p[NO_SNODECBR_P];
  for (int i = 0; i < NO_SNODECBR_P; i++) {
     cbrnode_p[i]=ServiceNodeCBR(n[0],
      wlink.GetNode(1+NO_HNODE+NO_UNODE+NO_SNODECBR+i),
      rPort[1+NO_HNODE+NO_UNODE+NO_SNODECBR+i],
      lPort[1+NO_HNODE+NO_UNODE+NO_SNODECBR+i],
      cbrrate_p[(i%3)], cbrpsize_p[(i%3)], cbr_para[(i%3)]);
  }

  //read energy consumption rate from ecr.txt
  ifstream ecr( "ecr.txt", ios::in);
  if (!ecr){
        cerr<<"File could not be opened"<<endl;
```

```
        exit(1);
}
double Si[NO_NODE];
double Sb[NO_NODE];
int nnum=0;
double temps;
while (!ecr.eof()){
        char temp[256];
        ecr.getline(temp,256);
if (temp[0]=='E'){
   char * pch;
   pch=strrchr(temp,' '); //get the pointer on last appreance of space.
                          //It will get the last token in the line.
   if (temp[3]=='i') Si[nnum] = atof ( pch );
        //convert string to double. consumption rate during idle
   else {
      Sb[nnum] = atof ( pch );//consumption rate at busy
      if (Sb[nnum]<Si[nnum]){
         temps=Sb[nnum];
         Sb[nnum]=Si[nnum];
         Si[nnum]=temps;
        }
        nnum++;
     }
   }
}
//an array used to record node's safe time
Time_t safeTime[NO_NODE];
for (int i = 0; i < (NO_NODE); i++) safeTime[i]=0;
//an array used to record nodes' previous status change time
Time_t pTime[NO_NODE];
for (int i = 0; i < (NO_NODE); i++) pTime[i]=0;
//an array used to record nodes' previous energy at last status change time
Energy_t pEnergy[NO_NODE];
for (int i = 0; i < (NO_NODE); i++){
   if (i<(NO_HNODE+NO_UNODE)) pEnergy[i]=INIT_ENERGY*2;
   else pEnergy[i]=INIT_ENERGY;
}
```

```
//Start call user scenario
gs->On(4);//allowed trace on layer 4
gs->On(3);//allowed trace on layer 4
gs->TimePrecision(3);//set bit number after dote
int activeu= 1;
static_cast<NodeSN*>(n[1+NO_HNODE+activeu])->getSNBattery()->
            SetRemainingEnergy(INIT_ENERGY*2);
RandomChoose t(n, cbrnode, cbrnode_p, n[activeu+1+NO_HNODE], Sb, Si,
                                safeTime, pTime, pEnergy);
t.Schedule(new TimerEvent, 1.5);
RecordDeath t1(n);
t1.Schedule(new TimerEvent, 1.0);
//s.Progress(1.0);
s.StopAt(SIM_TIME);
//s.AnimationUpdateInterval(Time("1ms"));
//s.StartAnimation(0, true);
s.Run();
gs->Close();
cout << "Simulation complete" << endl;
for (int i = 0; i < NO_NODE; i++)
        cout<<" node "<<n[1+i]->Id()<<" "<< static_cast<NodeSN*>
                (n[1+i])->getSNBattery()->GetRemainingEnergy()<<endl;
}
```

APPENDIX B

DETECTION ALGORITHM

This Appendix provides the code used for intrusion derection. The detection algorithm is the chi-suqare statistic test described in chapter seven. It is written in Java programming language.

DetectionAlgorithm.java

```
/* Written by Bo Zhou, Liverpool JMU, 2006
   Detect anomalies by reading event records and calculating the anomalous index
   X².
*/

import java.io.*;
import java.math.*;
import java.util.*;

public class DetectionAlgorithm {
     public static void main( String args[] )
    {
        File pDistribution = new File( "distribution.txt" );
        File mRecords= new File ("records.txt");
        //format of event record should be
      //service id, user id, action, time, duration, parameter, packet in, packet out
        int NO_SNODE=10;
        int S_TIME=1200;
int STEPS=10;
 int NONNUM=5;
 //difference between node number and last digital of IP addr
double DECAY=0.3;//decay rate
java.text.DecimalFormat   df2   =new   java.text.DecimalFormat("#.00");
java.text.DecimalFormat   df4   =
            new   java.text.DecimalFormat("#.0000");
```

```
if ( pDistribution.exists()&& mRecords.exists() ) {
    System.out.print(
        pDistribution.getName() + " exists\n" +
        ( pDistribution.isFile() ? "is a file\n" :
                                   "is not a file\n" ) +
        ( pDistribution.isDirectory() ? "is a directory\n" :
                                        "is not a directory\n" ) +
        ( pDistribution.isAbsolute() ? "is absolute path\n" :
                                       "is not absolute path\n" ) +
        "Last modified: " + pDistribution.lastModified() +
        "\nLength: " + pDistribution.length() +
        "\nPath: " + pDistribution.getPath() +
        "\nAbsolute path: " + pDistribution.getAbsolutePath() +
        "\nParent: " + pDistribution.getParent() +"\n\n");

if ( pDistribution.isFile() && mRecords.isFile()) {
        try {
            RandomAccessFile pdistribution =
                new RandomAccessFile( pDistribution, "rw" );
            RandomAccessFile   mrecords=
                new RandomAccessFile( mRecords, "r" );
            String record_e, record_p;//individual event records
            String[] detail_e, detail_p;
            String[] detail_IP;
            long[] node_p=new long[NO_SNODE];
            double[] X2=new double[3000];
                  //record all X2 to calculate mean and Sx2. i.e. threshold
            int nodeNum=0; //number of current activated service node
            int recordNum=0; //number of records has been processed
            int alarmNum=0; //recoed how many alarms have been raised
        int rareNum=0; //record number of rare events
            int falseNum=0; //record alarms raised without connection with
                                  // rare events.i.e. number of false alarms.
          //record starting position of each node's related records
             // in distribution.txt file
          while( ( record_p = pdistribution.readLine())!=null){
                if (record_p.startsWith("node")){
```

```
            node_p[nodeNum]=pdistribution.getFilePointer();
            nodeNum++;
        }
}
//process each record
while( ( record_e = mrecords.readLine() ) != null ){
        detail_e = record_e.split("\\s");//split recrod by spaces
    if (detail_e[2].equalsIgnoreCase("on")){
     double stime=Double.parseDouble(detail_e[3]);
     detail_IP = detail_e[0].split("\\.");
     int node_num=Integer.parseInt(detail_IP[3]);
     nodeNum=node_num-NONNUM;
      pdistribution.seek(node_p[nodeNum]);
      long tempStart=pdistribution.getFilePointer();
                                  //start position of off session
      record_p=pdistribution.readLine();
      detail_p=record_p.split("\\s");//split pdf by spaces
      double stime_min=Double.parseDouble(detail_p[0]);
      double stime_ele=Double.parseDouble(detail_p[1]);
      int elenum=(int)Math.round
                                    ((stime-stime_min)/stime_ele);
     String eleNum=Integer.toString(elenum);

     double ave=0.0;
     double currentQ=0.0;
     boolean match=false;
     boolean rare=false;

     for (int i=2; i<detail_p.length;i=i+3)
     {

       ave=Double.parseDouble(detail_p[i+1]);
            if (detail_p[i].equalsIgnoreCase(eleNum)){
detail_p[i+2]=Double.toString
   (Double.parseDouble(detail_p[i+2])*(1-DECAY)+DECAY);
if (ave==0) {
        ave=0.001;
        System.out.print("Rare event! ");
```

```
                rareNum++;
                rare=true;
            }
            match=true;
        }else detail_p[i+2]=Double.toString
                    (Double.parseDouble(detail_p[i+2])*(1-DECAY));
        if (ave!=0){
            currentQ=currentQ+Math.pow
                        ((Double.parseDouble(detail_p[i+2])-ave), 2)/ave;
        }
        }
        if (match==false){ //the eleNum apprears for the first time
                    ave=0.001;
                    System.out.print("Rare event! ");
                    rareNum++;
                    rare=true;
                    currentQ=currentQ+Math.pow((DECAY-ave), 2)/ave;
        }
        String tempString=detail_p[0]+" "+detail_p[1]+" ";
        pdistribution.seek(tempStart+tempString.length());
        for (int i=2; i<detail_p.length; i=i+3){
                    pdistribution.writeBytes(detail_p[i]+" "+detail_p[i+1]+
                     " "+df2.format(Double.parseDouble(detail_p[i+2]))+" ");
        }
        recordNum++;
        if ((currentQ>2.34)){//calculate from first time run this data set
                    System.out.print("Alarm! "+"X2="+df4.format(currentQ)+
                                        " Record num="+recordNum+"\n");
                      alarmNum++;
                    if(rare==false)    falseNum++; //this is a false alarm
                  }
        }
        else if (detail_e[2].equalsIgnoreCase("off")){
                    double stime=Double.parseDouble(detail_e[3]);
                    double ptime=Double.parseDouble(detail_e[4]);
                    int ipacket=Integer.parseInt(detail_e[5]);
                    int opacket=Integer.parseInt(detail_e[6]);
```

```
detail_IP = detail_e[0].split("\\.");
int node_num=Integer.parseInt(detail_IP[3]);
nodeNum=node_num-NONNUM;
pdistribution.seek(node_p[nodeNum]);
record_p=pdistribution.readLine();
long tempStart=pdistribution.getFilePointer();
                                    //start position of off session
record_p=pdistribution.readLine();
          //read twice to get PDs about 'off' related info
detail_p=record_p.split("\\s"); //split pdf by spaces
double ptime_min=Double.parseDouble(detail_p[0]);
double ptime_ele=Double.parseDouble(detail_p[1]);
int elenum=(int)Math.round((ptime-ptime_min)/ptime_ele);
String eleNum=Integer.toString(elenum);
double ave=0.0;
double currentQ=0.0;
boolean match=false;
boolean rare=false;

for (int i=2; i<detail_p.length;i=i+3){
    ave=Double.parseDouble(detail_p[i+1]);
    if (detail_p[i].equalsIgnoreCase(eleNum)){
       detail_p[i+2]=Double.toString
  (Double.parseDouble(detail_p[i+2])*(1-DECAY)+DECAY);
   if (ave==0) {
       ave=0.001;
       System.out.print("Rare event! ");
       rareNum++;
       rare=true;
     }
     match=true;
} else detail_p[i+2]=Double.toString
                    (Double.parseDouble(detail_p[i+2])*(1-DECAY));
    if (ave!=0){
        currentQ=currentQ+Math.pow(
                    (Double.parseDouble(detail_p[i+2])-ave), 2)/ave;
    }
}
```

```
if(match==false)//the eleNum apprears for the first time {
        ave=0.001;
        System.out.print("Rare event! ");
        rareNum++;
        rare=true;
        currentQ=currentQ+Math.pow((DECAY-ave), 2)/ave;
 }
 String tempString=detail_p[0]+" "+detail_p[1]+" ";
 pdistribution.seek(tempStart+tempString.length());
 for (int i=2; i<detail_p.length; i=i+3){
      pdistribution.writeBytes(detail_p[i]+" "+detail_p[i+1]+
         " "+df2.format(Double.parseDouble(detail_p[i+2]))+" ");
 }
 recordNum++;
 if ((currentQ>2.34)){// calculate from first time run this data set
      System.out.print("Alarm! "+"X2="+df4.format(currentQ)+
                                        " Record num="+recordNum+"\n");
      alarmNum++;
      if(rare==false)     falseNum++; //this is a false alarm
  }
 //calculate input packets
 pdistribution.seek(node_p[nodeNum]);
 record_p=pdistribution.readLine();
 record_p=pdistribution.readLine();
 tempStart=pdistribution.getFilePointer();
 record_p=pdistribution.readLine();
                        //read twice to get PDs about 'off' related info
 detail_p=record_p.split("\\s");//split pdf by spaces
 double ipacket_min=Double.parseDouble(detail_p[0]);
 double ipacket_ele=Double.parseDouble(detail_p[1]);
 elenum=(int)Math.round((ipacket-ipacket_min)/ipacket_ele);
 eleNum=Integer.toString(elenum);
 ave=0.0;
 currentQ=0.0;
 match=false;
 rare=false;
 for (int i=2; i<detail_p.length;i=i+3){
     ave=Double.parseDouble(detail_p[i+1]);
```

```
if (detail_p[i].equalsIgnoreCase(eleNum)){
      detail_p[i+2]=Double.toString(
        Double.parseDouble(detail_p[i+2])*(1-DECAY)+DECAY);
      if (ave==0) {
          ave=0.001;
          System.out.print("Rare event! ");
          rareNum++;
          rare=true;
      }
          match=true;
      }else detail_p[i+2]=Double.toString(
                Double.parseDouble(detail_p[i+2])*(1-DECAY));
      if (ave!=0){
            currentQ=currentQ+Math.pow
                  ((Double.parseDouble(detail_p[i+2])-ave), 2)/ave;
      }
}
if(match==false){//the eleNum apprears for the first time
      ave=0.001;
      System.out.print("Rare event! ");
      rareNum++;
      rare=true;
      currentQ=currentQ+Math.pow((DECAY-ave), 2)/ave;
}
tempString=detail_p[0]+" "+detail_p[1]+" ";
pdistribution.seek(tempStart+tempString.length());
for (int i=2; i<detail_p.length; i=i+3) {
       pdistribution.writeBytes(detail_p[i]+" "+detail_p[i+1]+
             " "+df2.format(Double.parseDouble(detail_p[i+2]))+" ");
}
recordNum++;
if ((currentQ>2.34)){// calculate from first time run this data set
      System.out.print("Alarm! "+"X2="+df4.format(currentQ)+
                              " Record num="+recordNum+"\n");
      alarmNum++;
      if(rare==false) falseNum++;   //this is a false alarm
}
```

```
  //calculate output packets
pdistribution.seek(node_p[nodeNum]);
record_p=pdistribution.readLine();
record_p=pdistribution.readLine();
record_p=pdistribution.readLine();
tempStart=pdistribution.getFilePointer();
record_p=pdistribution.readLine();
   //read more lines to get PDs about 'off' related info
detail_p=record_p.split("\\s");//split pdf by spaces
double opacket_min=Double.parseDouble(detail_p[0]);
double opacket_ele=Double.parseDouble(detail_p[1]);
elenum=(int)Math.round((opacket-opacket_min)/opacket_ele);
eleNum=Integer.toString(elenum);
ave=0.0;
currentQ=0.0;
match=false;
rare=false;
for (int i=2; i<detail_p.length;i=i+3){
    ave=Double.parseDouble(detail_p[i+1]);
    if (detail_p[i].equalsIgnoreCase(eleNum)) {
        detail_p[i+2]=Double.toString
  (Double.parseDouble(detail_p[i+2])*(1-DECAY)+DECAY);
    if (ave==0) {
        ave=0.001;
        System.out.print("Rare event! ");
        rareNum++;
        rare=true;
    }
    match=true;
}else detail_p[i+2]=Double.toString(
                    Double.parseDouble(detail_p[i+2])*(1-DECAY));
if (ave!=0){
        currentQ=currentQ+Math.pow
                    ((Double.parseDouble(detail_p[i+2])-ave), 2)/ave;
    }
}
if(match==false){ //the eleNum apprears for the first time
    ave=0.001;
```

```
        System.out.print("Rare event! ");
        rareNum++;
        rare=true;
        currentQ=currentQ+Math.pow((DECAY-ave), 2)/ave;
    }
    tempString=detail_p[0]+" "+detail_p[1]+" ";
    pdistribution.seek(tempStart+tempString.length());
    for (int i=2; i<detail_p.length; i=i+3){
          pdistribution.writeBytes(detail_p[i]+" "+detail_p[i+1]+
                " "+df2.format(Double.parseDouble(detail_p[i+2]))+" ");
        }
        recordNum++;
        if ((currentQ>2.34)){// calculate from first time run this data set
            System.out.print("Alarm! "+"X2="+df4.format(currentQ)+
                                        " Record num="+recordNum+"\n");
            alarmNum++;
            if(rare==false)    falseNum++; //this is a false alarm
         }
    } else if (detail_e[2].equalsIgnoreCase("set")){
        double stime=Double.parseDouble(detail_e[3]);
        double para=Double.parseDouble(detail_e[4]);
        detail_IP = detail_e[0].split("\\.");
        int node_num=Integer.parseInt(detail_IP[3]);
        nodeNum=node_num-NONNUM;
        pdistribution.seek(node_p[nodeNum]);
        record_p=pdistribution.readLine();
        record_p=pdistribution.readLine();
        record_p=pdistribution.readLine();
        record_p=pdistribution.readLine();
        long tempStart=pdistribution.getFilePointer();
        record_p=pdistribution.readLine();
        detail_p=record_p.split("\\s");//split pdf by spaces
        double para_min=Double.parseDouble(detail_p[0]);
        double para_ele=Double.parseDouble(detail_p[1]);
        int elenum=(int)Math.round((para-para_min)/para_ele);
        String eleNum=Integer.toString(elenum);
        double ave=0.0;
        double currentQ=0.0;
```

```
boolean match=false;
boolean rare=false;
for (int i=2; i<detail_p.length;i=i+3){
      ave=Double.parseDouble(detail_p[i+1]);
   if (detail_p[i].equalsIgnoreCase(eleNum)){
       detail_p[i+2]=Double.toString(
   Double.parseDouble(detail_p[i+2])*(1-DECAY)+DECAY);
       if (ave==0) {
           ave=0.001;
           System.out.print("Rare event! ");
           rareNum++;
           rare=true;
       }
       match=true;
}else detail_p[i+2]=Double.toString
     (Double.parseDouble(detail_p[i+2])*(1-DECAY));
if (ave!=0){
     currentQ=currentQ+Math.pow
                ((Double.parseDouble(detail_p[i+2])-ave), 2)/ave;
}
}
if(match==false){//the eleNum apprears for the first time
    ave=0.001;
    System.out.print("Rare event! ");
    rareNum++;
    rare=true;
    currentQ=currentQ+Math.pow((DECAY-ave), 2)/ave;
}
String tempString=detail_p[0]+" "+detail_p[1]+" ";
pdistribution.seek(tempStart+tempString.length());
for (int i=2; i<detail_p.length; i=i+3){
    pdistribution.writeBytes(detail_p[i]+" "+detail_p[i+1]+
            " "+df2.format(Double.parseDouble(detail_p[i+2]))+" ");
}
recordNum++;
if ((currentQ>2.34)){ //calculate from first time run this data set
     System.out.print("Alarm! "+"X2="+df4.format(currentQ)+
                                  " Record num="+recordNum+"\n");
```

```
                alarmNum++;
                if(rare==false)    falseNum++;   //this is a false alarm
            }
        }
}//end of while
pdistribution.close();
//save Q value in a file for later analysis
File Q_all = new File( "Q.txt" );
RandomAccessFile q_all =new RandomAccessFile( Q_all, "rw" );
q_all.seek(q_all.length());
//calculate false alarm rate
double far, hr;//far=false alarm rate //hr=hit rate
if (recordNum==rareNum) far=(falseNum+0.0)/1.0;
                                    //to avoid dividing 0 at denominator;
else far=(falseNum+0.0)/(recordNum-rareNum+0.0);
if (rareNum==0) hr=(alarmNum-falseNum+0.0)/1.0;
                                    //to avoid dividing 0 at denominator;
else hr=(alarmNum-falseNum+0.0)/(rareNum+0.0);
System.out.print("\nToatlAlarmNum="+alarmNum+
        " FalseAlarmNum="+falseNum+" RecordNum="+(recordNum)+
    " RareEventNum="+rareNum+" FalseAlarmRate="+df4.format(far)+
        " HitRate="+df4.format(hr));
q_all.writeBytes("\nToatlAlarmNum="+alarmNum+
        " FalseAlarmNum="+falseNum+" RecordNum="+(recordNum)+
    " RareEventNum="+rareNum+" FalseAlarmRate="+df4.format(far)+
        " HitRate="+df4.format(hr));
//calculate mean and Sx2
double totalX2=0;
double SX2=0;
double meanX2, Threshould;
int halfNum=recordNum/2;
for (int i=0; i<(halfNum);i++){
    totalX2+=X2[i];
}
 meanX2=totalX2/halfNum;
 for (int i=0; i<halfNum;i++){
     SX2+=Math.pow((meanX2-X2[i]), 2);
 }
```

```
    SX2=Math.sqrt(SX2/halfNum);
    Threshould=meanX2+3*SX2;
    System.out.print("\nmeanX2="+meanX2+" SX2="+SX2+
   }
  catch( IOException e2 ) {
       System.out.print("FILE ERROR"+"\n");
   }
}else if ( pDistribution.isDirectory() ) {
          String directory[] = pDistribution.list();
            System.out.print( "\n\nDirectory contents:\n");
          for ( int i = 0; i < directory.length; i++ )
            System.out.print( directory[ i ] + "\n" );
      }
    }
    else {
      System.out.print(   " Does Not Exist\n"+ " FILE ERROR\n");
    }
  }
}
```