Lappeenrannan teknillinen yliopisto
Lappeenranta University of Technology

Edited by Jouni Ikonen, Jari Porras and Pekka Jäppinen

3RD WORKSHOP ON APPLICATIONS OF WIRELESS COMMUNICATIONS

The proceedings of the 3rd Workshop on Applications of Wireless Communications

**Acta Universitatis
Lappeenrantaensis
211**

LAPPEENRANTA
UNIVERSITY OF TECHNOLOGY

Edited by Jouni Ikonen, Jari Porras and Pekka Jäppinen

3RD WORKSHOP ON APPLICATIONS OF WIRELESS COMMUNICATIONS

The proceedings of the 3rd Workshop on Applications of Wireless Communications

Acta Universitatis
Lappeenrantaensis
211

ISBN 952-214-057-0
ISBN 952-214-058-9 (PDF)
ISSN 1456-4491

Lappeenrannan teknillinen yliopisto
Digipaino 2005

MESSAGE FROM THE CHAIRS

Welcome to the Lappeenranta and to the 3rd Workshop on Applications of Wireless Communications. The workshop was established to bring together researchers, engineers and students working on the field of wireless communication. In 1799, Count Rumford identified engineering as "The application of science to the common purpose of life". Following Count Rumford definition, WAWC concentrates on the question how wireless communications can be applied in the different areas of life.

This year there were 10 submissions to the workshop. Four of the papers submitted were from Finland while the rest from Europe and Asia. These submissions were delivered to at least two program committee members for a blind review. In addition to that program committee chair reviewed all the submissions. As a result three submissions were accepted for the workshop, thus giving the acceptance rate of 30% for the workshop. The final program includes a keynote presentation from Professor Audestad dealing the effects of unwiring the society. Furthermore three invited papers are included to cover the gaming, business and user aspects of wireless world.

Although this is 3rd year that the workshop is arranged, this event has longer traditions as WAWC is arranged along with the 14th summer school on telecommunications. The summer school has successfully brought together researchers and companies for fruitful conversations for 14 years in a row, which itself shows the importance of the event. This year summer school concentrates on the Personal Area Networking starting from radio technology issues and ending to personal area networking aspects. These topics work as a good introduction for the WAWC presentations thus linking the workshop and summer school tightly together.

After the workshop the final event of summer school is a code camp. The camp is an intensive 24 hour coding, learning and competing event for students. During the camp participants program applications that this year rely on personal area networking. The groups are encouraged to interact with each other and solve problems together and give ideas to each other. All the creations and the group actions are evaluated and the best group is awarded a prize as recognition.

We hope you have enjoyable event in Lappeenranta and hope to see you here again next year.

Lappeenranta, 23.5. 2005

Pekka Jäppinen, Jouni Ikonen and Jari Porras

Organization

Summer School Chair

Dr.Tech. Jouni Ikonen, Lappeenranta University of Technology

WAWC Program Chair

Dr. Tech. Pekka Jäppinen, Lappeenranta University of Technology

International Program Committee

Professor Jari Porras, Lappeenranta University of Technology, Finland
Professor Jarmo Harju, Tampere Univerity of Technology, Finland
Jukka K. Nurminen, Nokia Research Center, Finland
Professor Olli Martikainen, Oulu University, Finland
Assistant Professor Denis Trcek, Institut "Jozef Stefan", Ljubljana, Slovenia
Assistant Professor Dario Maggiorini, Università degli Studi di Milano, Italy

Keynote speaker

Prof. Jan Arild Audestad, Norwegian University of Science and Technology and Gjøvik University College.

Organizing Chair

Professor Jari Porras, Lappeenranta University of Technology

Local Organizing Committee

Matti Juutilainen, Lappeenranta University of Technology, School Program Chair
Jani Peusaari, Lappeenranta University of Technology, Code Camp Chair
Janne Oksanen, Lappeenranta University of Technology, Network Arrangements
Harri Hämäläinen, Lappeenranta University of Technology, Web-pages
Kimmo Koskinen, Lappeenranta University of Technology, Technology strategist
Arto Hämäläinen, Lappeenranta University of Technology, Documentation

Table of Contents

THE UNWIRED SOCIETY: FLEXIBLE AND ROBUST BUT DANGEROUSLY VULNERABLE

Jan A Audestad
Senior Adviser, Telenor Corporate Management,
Adjunct professor Norwegian University of Science and Technology (NTNU) (telematics),
Adjunct professor Gjøvik University College (information security).

ABSTRACT

The evolution of telecommunications and information technology since 1995 has led to enormous flexibility for users, efficient production of goods and services for manufacturers, and efficient management of society. The price we may have to pay for this development is that we have created a society that is vulnerable to certain types of malicious attacks. What makes it worse is that the evolution has been irreversible.

About five years ago investigations of the internet, the web and several biological and social networks revealed that many of these networks had a structure that made them robust against random destruction of nodes but very vulnerable to attacks directed at certain nodes called hubs. Such networks were named scale-free networks. Since then, the structure of these networks and how they may form have been subject to intense mathematical study.

The paper shows models of computer networks that may belong to the class of scale-free networks. Many researchers believes that this is the case but this has not been definitely confirmed because it is tremendously difficult to reveal the complete structure of internet, email networks and the web because of their size, the dynamic changes going on all the time, and lack of adequate experimental methods.

The number of devices containing CPUs is more than 1000 billion devices, where only 1 billion of these devices are under direct human control. All the devices are connected to the internet either directly or indirectly and makes up the vast, connected computer infrastructure of the world. This gives rise to unprecedented security challenges when designing these devices and the way in which they interact.

1. UBIQUITY AND VERSATILITY OF COMPUTING

It is assumed that there are about 1000 billion devices on Earth containing CPUs. Only about one billion of these devices are what we traditionally call a computer (PC, server or mainframe). Most of these CPUs are doing autonomous tasks not directly controlled by humans.

Autonomous CPUs are found in automobiles, aircraft, measuring equipment, internet routers, machines, control systems, water and sewage pumps, sluices, valves in oil refineries, mobile phones, smart cards, printers, computer mouse, sensors of all kind, and so on and so forth. Usually it does not occur to us that these devices exist, and we recognise them only if the car stops for inexplicable reasons or the automatic teller machine is out of order when we need cash.

Almost all the CPUs are connected to the internet either directly or indirectly. It is then possible in principle, though very difficult if exact address information is not known, to reach any such CPU from any other CPU. The CPUs are then making up a formidable computing infrastructure connecting every corner of the globe.

Most CPUs are contained in small devices capable of doing just a few dedicated tasks. Many of the CPUs are contained in mobile devices. An increasing number of them, also those in fixed locations, communicate via radio interfaces. Radio connections exist for example between the mouse and the PC, between the PC and the printer, or between pressure sensors in the tyres and a computer in the car.

RFIDs containing CPUs will complicate this picture manifold when we find out how to provide them with enough energy to accommodate a CPU rather than just simple electronics.

Every activity in society is controlled by computers. Management of society and industry, interaction between people and the authorities, commerce and banking, retrieval and dissemination of information and communication between people are part of the computer infrastructure under direct human control. The operation of the infrastructure of society (transport, energy distribution, and telecommunications) and the production processes in industry are largely autonomous systems. In fact, all human controlled activities rely also on an autonomous infrastructure – the internet and local networks and computer systems!

This computerisation of society has mainly taken place after 1995 and has been an irreversible process: it is impossible to revert to old routines if the new ones fail. In fact, the "ICT-ation" of society – technical or administrative – is an irreversible process.

In this paper I shall not consider the development of radio interfaces and devices containing transmitters and receivers which I believe will be the main theme of this workshop. What I shall do is to describe the big picture and from that derive some characteristics that need to be taken into account when new systems are designed and implemented in the existing computer infrastructure.

2. THE BIG PICTURE

Let us start with drawing the big picture. We need to understand this picture in order to see the impact innovations even in the small and local scale may have on the information and communication infrastructure at large.

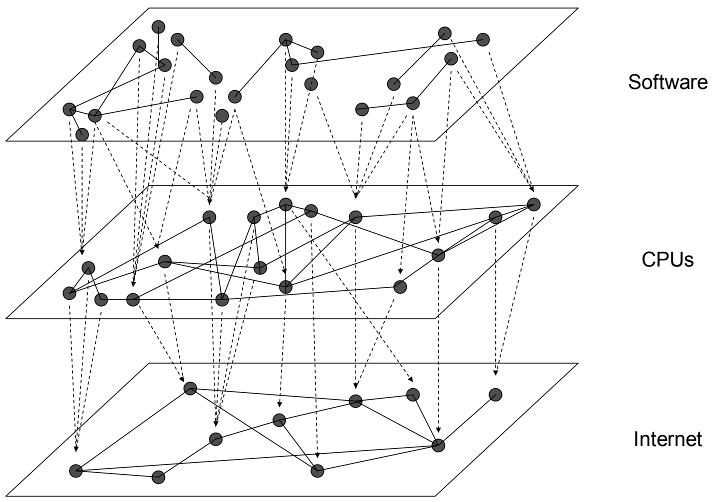


Figure 1 The big picture of the computing infrastructure

Figure 1 shows a layered structure consisting of three networks. Internet, including local systems such as LANs and piconets, ensuring the capability of passing bits from one location to another, is at the bottom of the structure. A node in this network (black dish) represents a router or another device responsible for passing bits. A link between two nodes means that a direct communication channel exists between the devices.

The CPUs reside on the layer above. The CPUs are nodes in this network and there is a direct link between two CPUs if they ever take part in a common computational task (for example exchanging email). The links may even be weighted where the weight represents how often the CPUs take part in a common activity. Different interrelationships such as email, web search and distributed processing may be described in terms of separate graphs. The different sub-graphs constructed in this way may have properties that are important for assessing vulnerability, security threats or traffic handling capacity of the computer infrastructure of society.

One important observation is that there exist paths between CPUs that are not directly linked. This means that it is possible for two CPUs to exchange information via intermediary devices. This is in fact how malicious code such as viruses and worms are spread in internet.

The CPUs are connected to nodes in the internet graph as shown but the structures of the two graphs are independent of each other.

The CPUs are the kernel of computing devices containing from a few to millions of executable programs or documents. A program or document can be represented as nodes in a graph while links indicates that the program or document is used in a common computation session. Though programs and documents reside in computers with CPUs, this is a graph structure independent of the CPU network below.

The structure shown in Figure 1 is not static but highly dynamic in a stochastic way. The interconnectivity graphs of CPUs and software not only grow in size at a rapid rate but also changes internal structure. This stochastic behaviour is in a way fractal since it is likely to contain a continuum of timescales from seconds to years. We do not know this for sure but experience and intuition tells us that it must be so – and that this dynamics is completely different from the dynamics of the telephone network and the GSM network. These networks are planned to minute detail; the computer networks just evolve.

This is the big picture of the computing infrastructure. Let us then see how this infrastructure may have grown from virtually nothing.

3. NATURAL GROWTH OF NETWORKS

Nature, society and technology are full of networks that started developing from a small seed. Examples of such networks are cell metabolism where substances taking part in the same chemical reaction are linked, co-authorship in science, industrial ownership, and the World Wide Web and the internet.

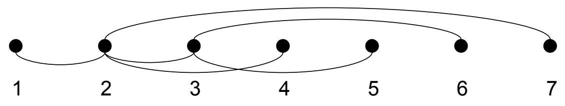


Figure 2 Growing a network

Figure 2 shows how such an undirected graph may evolve[1]. The numbers refer to the generation of the node. One algorithm among several is as follows. Call the number of links terminating at a node the *degree* of that node. Start with interconnected nodes 1 and 2 and then apply the following algorithm. At generation n add node $n + 1$ and link it to one of the previous nodes in such a way that the probability of choosing a particular node is proportional to the degree of that node. The degree distribution for a large graph grown in this or similar ways is $P(g) = 1/g^{\gamma}$ where γ is a constant. This distribution is called a Pareto distribution or power-law distribution. In the particular example, $\gamma = 2$. In sharp contrast, the node degree of a graph where links between nodes are equally probable, the degree will be Poisson distributed with almost all nodes having a degree very close to the average of the distribution. Networks with degree distributions following a power law are called scale-free networks because the degrees are not concentrated narrowly around a mean value (or single "scale"). One important observation is that the tail of the distribution is thick (or the kurtosis or fourth order moment of the distribution is large), which means that the probability of finding

[1] Graphs may also be directed where an arrow on a link indicates that one node is connected to another node but not vice versa. The web is such a graph. Directed graphs may grow in similar ways as undirected graphs.

occurrences far away from the average is much larger than in Poisson or Gauss distributions where the tail drops off at exponential rate.

Scale-free networks were discovered and analysed as late as 1999 (Albert and Barabási). Since then, a number of properties of naturally grown networks have been uncovered. One particular aspect is vulnerability. Another contradictory observation is that these networks are unusually robust against random destruction. This is why nature is so full of them.

4. VULNERABILITY AND ROBUSTNESS

Because of preferential growth, some nodes of the network will grow to become very large with links to very many other nodes. Such nodes are called *hubs*. In the World Wide Web, the search engines are obviously hubs and so are web pages containing own search engines or many hyperlinks to other pages. In the internet, huge routers and certain functions in the network, for example inter-traffic interworking units, are hubs. Investigations of these networks indicate that they are scale-free but this is far from being definitely confirmed; however, the structure is certainly one satisfying a thick-tail distribution over a large range of the random variable.

A very small scale-free graph is shown in Figure 3. If nodes in this graph are removed at random, the probability that the removed node has a large edge degree is small. This means that in a huge graph such as the internet or the CPU network very little damage will be done to the connectivity of the graph even if many nodes are removed at random. In the figure, 4 nodes (25% of all nodes) are removed at random and the graph is still connected. On the other hand, if the attack on the graph is directed towards the hubs just a few nodes need to be removed before the whole graph falls apart. In the figure two hubs are remover and the network disintegrates completely. This effect is particularly noticeable in the WWW: remove the search engines and the web falls apart.

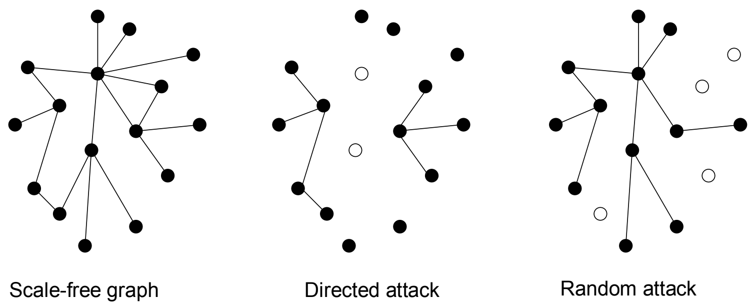


Figure 3 Scale-free networks

Another observation concerning scale-free networks is that the infection threshold of these networks is zero. This needs some explanation. In an epidemic network where the link distribution is Poisson, a certain number of nodes (people) must be infected before the disease starts to spread. This is why random vaccination works. In a scale-free epidemiologic network such as that of AIDS, no such threshold exists: the whole graph will be infected if a single node is infected. The time it takes to infect the whole network just depends how long it takes before major hubs are infected. As soon as a hub is reached the disease spreads quickly. In scale-free networks random vaccination does not work: the spreading can only be stopped by identifying and vaccinating the hubs – in the AIDS epidemic this is an ethic challenge. From an information security viewpoint, this also means that data viruses are best stopped by protecting the hubs better than the smaller nodes. The spreading of the internet worm SoBig a couple of years ago was slowed down in this way because many large companies had effective routines for filtering out emails containing suspicious attachments.

Since our intuition and current studies tell us – though we have no definite proofs that this is true – that the internet and its major application (email and web) are scale-free networks, we may conclude that it is extremely easy to contaminate these networks with data viruses and worms if you first can figure out how to do it. The problem is that so much of the society depends on ICT that a major virus attack may put important functions of society out of action either by disturbing communication or by destroying critical computer programs.

5. PROTECTION

We can, of course, protect us against many attacks by malicious software. We may install firewalls stopping doubtful software and accesses by parties we do not know. There are one mathematical and one human reason why this does not work out properly.

The mathematical reason has to do with a deep result in mathematical logic called the undecidability of the Turing halting problem. Turing’s theorem states that it is not possible to design a general algorithm that can answer yes or no to the question whether another arbitrary algorithm will, for an arbitrary input, return an answer to the problem in finite time or just continue searching for a solution indefinitely. It is easy to show that if we were able to construct a virus detection program that could detect every existing and future virus code, this would violate Turing’s theorem. Hence, however sophisticated the firewall is, it does not protect us against future virus attacks.

The human reason is that we want – and in fact need – to be a component of the vast network structures I described in section 2 in order to do our daily work or function socially in certain relationships. This means that we are part of not just one but several finely woven webs of interdependence that certainly not are simple random networks but more likely are networks with scale-free properties. This makes us vulnerable to several types of attack and some of these attacks may also utilise the fact that scale-free networks do not have contamination thresholds. One of the most effective protections we have had so far is obviously the complexity of the networks. It is not just hard to construct countermeasures but also to construct the weapons. On the other hand, we have no single idea about what the arsenal of information weapons may contain. There are organisations even in democratic countries that have an interest in building up such a secret arsenal.

A deeper understanding of the topologies of the technological and societal networks and their interdependence may help us not only to avoid attacks but also to design fault tolerant systems. These are systems that can recover by themselves after severe faults have taken them down entirely or partially and that sometimes also can deliver services while under attack. This brings information security into the field of fault tolerance and not only fault avoidance where firewalls and access control is at focus. The marriage of fault avoidance and information security is still a pristine research area.

6. UNWIRING THE SOCIETY

Figure 4 shows another view of the network (for example the internet). The network consists of a fixed kernel of routers. The network is dynamic because the routing of individual packets in the network is almost arbitrary: the routers push them, to their best effort and with a minimum of computation, toward the termination. The periphery consists of the CPUs shown in a different way in Figure 1. Some of these CPUs are located at geographically fixed points; but an increasing number of CPUs are mobile capable of accessing the kernel at arbitrary access points and also other fixed or mobile CPUs directly. Examples of such mobile CPUs are mobile phones, smart phones, smart cards, electronic wallets, personal computers, personal digital assistants, mobile hard disks, GPS receivers, MP3 devices, processors in automobiles and aircraft, and so on and so forth.

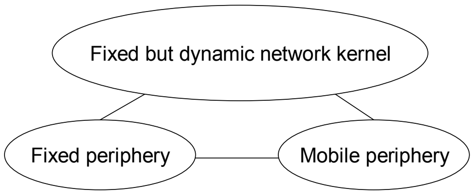


Figure 4 Telecommunications network model

Mobility provides us with enormous flexibility, and more and more businesses abandon fixed telephones and fixed personal computers: a good day's work is no longer the equivalent of sitting in front of the office desk from 0800 in the morning until 1600 in the afternoon. Now any time of the day at any place in the world may be office time. The result of this evolution is that the fixed periphery becomes smaller while the mobile periphery increases.

Furthermore, one person possesses not just one mobile device but a number of them that need to communicate with one another from time to time. In this way, a much richer structure of interfaces between devices grows up. These interfaces require new protocols and new platforms supporting distributed computing, mobility and resource sharing. This is good for flexibility but is a nightmare for information security.

7. LIFE AT THE PERIPHERY

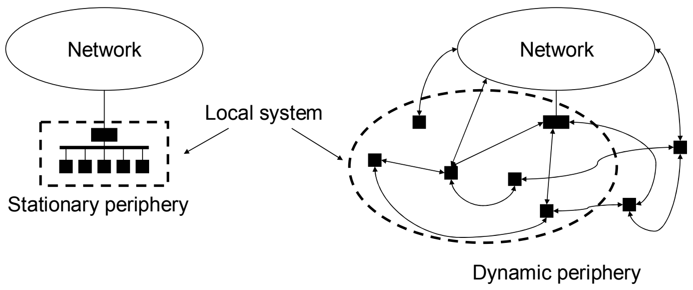


Figure 5 From stationary to dynamic access

Figure 5 illustrates how computer networks have developed during the last few years. The black rectangles represent computing devices of one kind or another – these are the nodes of the CPU graph in Figure 1. Previously, the local network was connected to the network via servers in the local network. All aspects of computation in this structure were under the authority of one organisation – the owner of the local network. Since then, the local network has become dynamic, mobile and distributed: computers within the network can access other computers of the local system from access points outside the local system; computers within or outside the local system may communicate directly without going through network interfaces or servers; computers may move between access points both within and outside the local system. The different computers taking part in the information exchange are of many different types covering a large range of memory size, computational speed and capabilities.

The local system is not just dynamic but it is also incredibly complex. In the stationary local system, communication both within the local system and with computers in other systems took place via servers. The owner of the local system could then enforce security policies and access control on these servers. This is only partially possible in the dynamic systems. These systems still contains servers for special purposes such as email and web access. Security policies are enforced on these systems as before. However, direct communication between devices using Bluetooth, infrared, WLAN and other interconnection methods opens up for uncontrolled exchange of information by circumventing the servers and violating the access control policy. This we can do without anyone discovering it.

The main driving force, in addition to being fun to do research and invent new things, behind developing all these access technologies is the need for more flexible systems that increase productivity and the output by each employee, reduce production cost, and do things that could not be done before.

The flexibility has multiplied the number of ways the computing infrastructure can be attacked. If it is impossible to send malicious software through the server, it may be delivered to a PDA over Bluetooth. When the PDA is synchronised with a personal computer on the inside of the firewall, the malicious software may be spread unnoticed because usually there is no access control between the PDA the PC. In fact, there is malicious software that is spread in this way via smart-phones and PDAs to personal computers. So far, such attacks have not been very successful probably because it is difficult to do it, but there is no comfort in this.

Enforcing security policies on lightweight interfaces and in simple equipment is not easy. The device may be too small to accommodate such functions or the implementation of them may be too expensive. Furthermore, there is a growing demand for being anonymous and untraceable on the internet but at the same time accountable for the transaction one takes part in. These are extremely difficult problems but there is much research going on in the field, and there is still food for numerous PhDs on the subject.

The dynamic periphery brings up several difficult problems in different areas such as:

- design of radio subsystem, device hardware and software, and protocols
- device configuration, resource management and protection of critical functions (for example tamper-resistant hardware for storing of secret information and encryption keys)
- networking issues such as autonomous creation of network topologies, presence detection and identification of device, attachment to friendly devices and prevention of malicious attachment, interference and manipulation
- device characteristics and user profiles for automatic setting of access control parameters and enforcement of user rights and security functions; this is one area where much is still to be done

All the above points involve information security in one way or another. These are problems concerned with the individual device. What we saw above was that the peripheral devices (or CPUs) are interconnected in voluntary and involuntary network structures. Ideally we only want voluntary interconnection between devices but the network formation at the CPU level is such that the device is involuntarily interconnected with all devices in the computer infrastructure. This configuration cannot be avoided.

Protecting the periphery against malicious attacks or accidental faults is not necessarily the same as protecting the network. Ideally, of course, if the peripheral devices are perfectly protected, the network itself is seemingly well protected. This is not true because of the complex interaction between the devices may cause network failures that cannot be traced back to single devices. These faults are structural but not localisable.

Furthermore, for mathematical (Turing's theorem) and economic reasons it is not possible to protect the devices against all possible malicious interference in the future. Again because of the scale-free (or at least thick-tailed) structure of the networks of Figure 1, even a small disturbance starting at a single peripheral device may cause structural collapse of the network.

However, the better we make the devices, the smaller is the chance that structural collapse will take place. Therefore, the security problems associated with all the small devices surrounding us must not be taken lightly.

REFERENCES

A.-L. Barabási, *Linked: The New Science of Networks*, Perseus Publishing, 2002

G. J. Chaitin, *Exploring Randomness*, Springer, 2001

D. L. Clark, *Enterprise Security: The Manager's Defence Guide*, Addison Wesley, 2003

Complexity, Vol. 8/No. 1, September/October 2002, Special issue: *Networks and Complexity*

S. N. Dorogovtsev and J. F. F. Mendes, *Evolution of Networks: From Biological Nets to the Internet and WWW*, Oxford University Press, 2003

K. Gaarder and J. A. Audestad, Feature Interaction Policies and the Undecidability of a General Feature Interaction Problem, *TINA93 Workshop*, L'Aquila, Italy, 27-30 September 1993

B. E. Helvik, Perspectives on the dependability of networks and services, *Telektronikk*, No 3, 2004

B. E. Helvik, *Dependable Computing Systems and Communications Networks: Design and Evaluation*, Draft Lecture Notes, Department of Telematics, NTNU, 2001

J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979

A. J Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997

H. Pham (*Ed*), *Handbook of Reliability Engineering*, Springer, 2003

J. E. Savage, *Models of Computation: Exploring the Power of Computing*, Addison-Wesley, 1998.

PUBLIC SCREENS AND PRIVATE MOBILE PHONE SCREENS IN MULTIPLAYER GAMES

Riku Suomela
Nokia Research Center, P.O. Box 100,
33721 Tampere, Finland
riku.suomela@nokia.com

ABSTRACT

The mobile phone is a gaming platform that is always available for the player. The phone is carried in the pocket, and whenever there is spare time, a game can be played. Phone owners encounter large public screens in many places, such as TVs in cafeterias. These screens could be used as a common public screen in mobile multiplayer gaming. The mobile phone has a private personal screen, whereas the public screen would serve all players in the game, as well as all the spectators in the place. We have an ongoing research project that study how public screens can be used in multiplayer games played with mobile phones. We have developed three games using this approach, and this paper looks at the issues in such games. We present three games, and analyze ingredients in games with a mobile phone and public screens.

KEYWORDS

Mobile multiplayer games, public screens, mobile phones, private information, public information

1. INTRODUCTION

Mobile phones have become the standard means for person-to-person communications. There are many ways people can communicate with each other, but mobile phones have one significant advantage over others - they are carried with their owners making them available wherever and whenever. The mobile phones are not just communication devices, as they can run general applications as well. The phones are also becoming open platforms, and third parties can develop software for them, which increase the speed of new application development. Web browsing and games are popular forms of applications in phones. These applications can provide entertainment when the user has some extra time.

Mobile phone owners use the phone at varying use conditions. In many cases, the users are in front of large public screens, which could be used as an extension to the mobile phone screens. The screen size of a mobile phone is typically small compared to public displays, and it would be beneficial to use the large screens in public spaces to allow more information to be shown.

The large screen could be used in any way, but there are some things that should not be done. The mobile phone screen is private, and it should not be shared with others unless the user wants this. Also, public screens are meant to serve more than one person at a time, so a single person should not be able to steal the screen.

Nomadic Media, an ITEA project we are working on, deals with public screens in public places among other things. In this paper, we look at public screens, and how they can be used in mobile multiplayer games. Here, we use the term public screen to refer to screens that are in public spaces and viewed by many people at the same time. If a user is using a screen at home, it is a private screen if it is not shared with others.

We have developed three games that use the public screen and the private screen in order to provide an enhanced gaming experience. All the players have a private screen at their use, which shows information only available to one player, whereas the public screen shows data that is visible to all the players. The games are based on two main design drivers: public information, and asymmetric information [1]. We have implemented a public screen component to the Multi-User Publishing Environment (MUPE) [2], to enable easy development of these games. The public screen component allows any MUPE application to easily use one or many public screens.

This paper is organized as follows. First, we look at the mobile games and games with public screens. Second, we present three games that we have developed, including a brief look at the technology used in development, followed by analysis of the public screen in the games. Finally, future work is discussed after which the work is concluded.

2. MOBILE MULTIPLAYER GAMES

Mobile multiplayer games differ from the desktop games in many regards. They are not played at a fixed location, they almost always use batteries restricting processing power and operation time, and the environment poses some problems for the use of the device, to name a few examples. Further, outputting and inputting information varies between devices, and the latency of the network connection is typically high, and varies a lot. Mobile games should be designed to keep these facts in mind.

A lot of research on mobile games has concentrated on using the real world as gaming arena. One of the earliest such games is geocaching played with a GPS receiver [3]. Pirates! [4], ARQuake [5], BotFighters [6], Can You See Me Now? [7], and Human Pacman [8] all use the real world as a gaming arena. In all games, the environment is a core part of the game. Sotamaa studies in his excellent article on BotFighters [9] how the real world can influence the gameplay. Tibia Micro Edition (TibiaME) [10] is a mobile multiplayer game in a fantasy setting, which is one of the first mobile multiplayer commercial games on a mobile phone.

Multiplayer game on a public screen is not a new idea. All the major consoles by Nintendo, Microsoft, and Sony allow many people to play in front of a public screen. Normally, the controllers have no display and thus no private screens, but there are some exceptions. Sega Dreamcast [11] had a Visual Memory Unit (VMU), a memory card with a small display and a controller. The VMU could be removed from the console, and some tasks could be made with the unit. Later, Nintendo [12] presented games that combined the Gamecube console and Gameboy Advance handheld console. Several games, such as The Legend of Zelda: Four Swords Adventures [13] take advantage of the public and private screens available.

Both screens should be used. Public screen gives information to all players, whereas private screen only to oneself. This should be used fully in the game design, as this gives a lot of possibilities for the designer. This is the main advantage of games with public and private screen.

Switching between the two screens. If the players need to constantly switch between which of the two screens they are viewing, this becomes strenuous. The game should be designed to require only a little switching between the screens. Intuitive controls on the mobile phone allow the device to be used without looking at it.

Spectators. In public screen games, there can be other people watching the game, and the content should be interesting for spectators. If only the players know what is going on in the game, there is no real spectator value, and the public screen only serves the players.

Lack of mobility. When the game is played in a static location with a public screen, the game lacks mobility. This is a serious drawback for mobile phone games, if they become location dependant. The game loses all benefits of the mobile platform, and this is the most serious drawback of designing such games. This can be overcome, if it is possible to design a game so, that it is not necessary to play with a public screen. Public screens should add something new and fun to the game, perhaps making it more interesting or increasing its replayability. Another option is to use multiple public screens in many locations, which increases the mobility, as the game is not restricted to a single public screen in a single location.

5. FUTURE WORK

All the games presented here set players against each other. The goal of each game was to beat others, and the public screen was used to show the game area, and each players units. This is a very limited approach, and a lot more variance is needed. The players should be playing in teams, all against the public screen, or the public screen would not be used to show the play arena.

We are conducting usability evaluation for the FirstStrike game. It can be played both with and without the public screen, and it will be interesting to see if the public screen really adds something to the game. The public screen allows the players to be identified with their photos, and the game is also slightly affected whether the public screen is present or not.

We are implementing public screen opponents to our co-operative game MupeDungeon. The public screen will contain tougher opponents for the players. The idea is to locate public screens in the real world, to open up new challenges in the mobile game.

We are also interested in using many public screens in a game. This would add mobility to a game, as a specific location would not be required to play a game. Locations, with public screens would add new content and experiences to the normal mobile game.

6. CONCLUSION

In this paper, we presented our ongoing research on public screens and mobile multiplayer games. We have designed and implemented public screen games, and three games were presented in this paper. The main design driver for each game was in information distribution between two information channels: public screen visible to all players, and private screens visible to players.

All games demonstrated a very different kind of gameplay. Racing Manager was a simulation game on the public screen, and the players could affect the outcome with their actions on the mobile phone. Bizarre Creatures used the public screen as the game arena and the mobile device as an advanced controller. FirstStrike could be played either with or without the public screen.

We analyzed the games and presented a list of desirable and undesirable features of mobile games using a public screen. The games should use both screens, but not require too much switching on which screen should be viewed. Further, the spectators, who are not actively participating in the game, should enjoy watching the game on screen. Finally, a mobile game should not be tied to a location with the public screen if possible, as it loses mobility in this case.

Acknowledgements

This work has been funded by the Nomadic Media ITEA project. We would also like to thank the Racing Manager, Bizarre Creatures, and FirstStrike teams for the implementations.

REFERENCES

[1] S. Björk, J. et al.:. Patterns in Game Design. (2005) Charles River Media, ISBN 1-58450-354-8
[2] Suomela, R., et al.: Open-Source Game Development with the Multi-User Publishing Environment (MUPE) Application Platform. Proceedings of the Third International Conference on Entertainment Computing (2004), Lecture Notes in Computer Science Vol. 3166 Springer (2004) 308-320.
[3] Geocaching. http://www.geocaching.com/
[4] Björk, S., et al.: Pirates! - Using the Physical World as a Game Board. Proceedings of the Human-Computer Interaction INTERACT'01 (2001) 423-430
[5] Thomas, B., et al.: ARQuake: an outdoor/indoor augmented reality first person application. Proceedings of the Fourth International Symposium on Wearable Computers. (2000) 139-146
[6] It's Alive: BotFighters. It's Alive (2001)
[7] Can You See Me now? http://www.canyouseemenow.co.uk/
[8] Cheok, A.D., et al.: Human Pacman: a mobile, wide-area entertainment system based on physical, social, and ubiquitous computing. Personal and Ubiquitous Computing. vol. 8. no. 2. Springer-Verlag (2004) 71-81
[9] Sotamaa, O. All The World's A Botfighter Stage: Notes on Location-based Multi-player Gaming. Proceedings of the Computer Games and Digital Cultures Conference, Tampere University Press (2002) 35-44
[10] CipSoft: Tibia Micro Edition (TibiaME). T-Mobile. (2003)
[11] Sega Dreamcast. http://www.sega.com/home.php
[12] Nintendo Gamecube and Gameboy advance. http://www.nintendo.com/home
[13] The Legend of Zelda: Four Swords Adventures for the Nintendo Gamecube. http://www.zelda.com/fourswords/launch/index.html
[14] Java 2, Micro Edition (J2ME) Wireless Toolkit 2.2. http://java.sun.com/products/j2mewtoolkit/download-2_2.html

[15] Nokia Open Source License Version 1.0a (NOKOS License). Available online at http://www.opensource.org/licenses/nokia.php
[16] MUPE website. http://www.mupe.net

ON THE EFFICIENT IMPLEMENTATION OF INSTANT MESSAGING SERVERS FOR WIRELESS USERS

Simone Leggio, Markku Kojo
Department of Computer Science
University of Helsinki, Finland
{simone.leggio, markku.kojo}@cs.helsinki.fi

ABSTRACT

Instant Messaging (IM) is one of the current killer applications in the Internet. Until now, IM services have been accessed mostly from wired networks; however, the number of wireless users is quickly growing. Wireless links have characteristics different from their wired counterparts; they have lower bandwidth capabilities, increased unreliability, variable delays and packet drops. IM servers have not been optimized for serving users accessing from wireless links. We have emulated IM sessions with a user accessing an IM server over a wireless link to study how the server behaves in a wireless environment. The experiments revealed shortcomings in IM server implementation and served as basis for giving guidelines for an efficient implementation of IM servers. This paper presents the guidelines and a high level timeout based, application level algorithm aiming to control the pace at which instant messages are pushed into the network from an IM server.

KEYWORDS

Instant Messaging, Wireless Links, Jabber, experiments, efficient data delivery.

1. INTRODUCTION

For several years Instant Messaging (IM) has been a very popular application; nowadays, millions of users make use of IM services every day, mostly accessing over wired links. The basic IM service, consisting in exchange of relatively short textual messages, is not problematic in such a network environment due to the reliability characteristics of the physical medium and transport protocol used to carry instant messages (usually TCP).

IM servers and protocols have been designed to exchange messages over wired links; however, the number of users that engage in IM conversations with remote parties utilizing a wireless device is growing, due to improvements in the mobile device capabilities and increase in the bandwidth of the wireless access links (GPRS/GSM 2.5 networks). Despite such enhancements, resource availability and capabilities of mobile devices and wireless links cannot equal those of wired links and stationary devices. In order to efficiently deploy an IM platform in networks involving wireless links, a number of issues must be considered to cope with the limited resources of wireless links and mobile devices.

Little earlier work exists on studying the behavior of the Instant Messaging in a mobile and wireless environment; in [6] the focus is on ensuring user mobility during IM sessions, rather than message exchange optimization. The authors propose a server based architecture for ensuring heterogeneous access to IM systems from several clients.

In this paper, we have set up an IM platform to experimentally study the behavior of IM message delivery in a wireless environment and to find out how the message delivery from an IM server could be enhanced. In wireless environment, in fact, the available bandwidth is often very low and packets delivered over the wireless link can get delayed or even totally lost due to handovers and impairments of the medium. The server must be prepared to handle efficiently these situations and limit the negative effect that the scarce resources of the wireless link bring about in an IM session. We focused on optimizing the way the IM server interacts with the TCP protocol to guarantee efficient and timely delivery of instant messages to the wireless user. Optimization of the IM protocol itself was out of the scope of our work. Based on the experiments, we present guidelines for the efficient implementation of an IM server and an algorithm that IM servers can use for regulating the pace at which IM messages are injected into the network.

The IM platform used in the experiments is Jabber [2], open source XML based IM and Presence system. The reasons for this choice are various: first of all, the platform is open source, which allows more operational freedom. Secondly, the XMPP protocol, which is based on the original Jabber protocol, is in the standardization phase in IETF [8,9]. However, the guidelines and the algorithm we provide are not Jabber specific, but they are applicable to any IM server and, in general, to any server in any network environment performing store-and-forward types of operations and employing TCP as the transport protocol. We stress anyway that the guidelines and the algorithm are especially important in wireless environments.

The rest of the paper is structured as follows: Section 2 presents our test arrangements, explaining the methodology used in the experiments; Section 3 gathers the most important lessons learnt on the experiments. The results are the basis for the guidelines for efficient implementation of IM servers, given in Section 4. We propose in Section 5 the timeout based algorithm for controlling the pace of message delivery into the network. Section 6 concludes the paper and discusses future work topics.

2. TEST ARRANGEMENTS

The Jabber IM platform is conceptually similar to the architecture of an e-mail system. Jabber clients are registered to a Jabber server, and they rely on the server they are registered to for forwarding instant messages to the recipient. The server analyzes a received instant message and checks the identity of the recipient client. If the recipient is registered to the domain the server is responsible for, the message is delivered to the recipient. If the recipient is not registered, the message is forwarded to the responsible server, and then from it to the intended recipient. Jabber messages are carried over TCP.

Figure 1 shows the test set-up and the wireless environment emulated in the experiments. We have analyzed Jabber message exchanges between a client that is connected through a wireless link and one or more clients accessing over a wired link. The fixed clients and the Jabber server are arbitrary hosts in the Internet. The Jabber server in use was jabberd 2.0b3 [3] and the clients in use were GAIM [4].

The aim of the tests was to study the effect that packet delays and losses provoke to a Jabber session; wireless links, together with user mobility, are very likely to cause such phenomena. The experiments were run to understand the IM behavior in the first place, not to gather measurement data for quantitative performance analysis. Therefore, the test runs were not repeated for a large number of times.

We emulated scenarios that could be possible in a real life situation where one of the clients in a Jabber session is attached over a wireless link. In all test cases, the clients connect to the same server. Situations implying inter-server communication were not studied. A typical configuration for the tests was a one-to-one communication between clients; however, some of the test cases were repeated using two source clients.

The basic idea behind all experiments was to have a flow of messages towards the wireless client (downlink direction) from a fixed client. The messages are first gathered at the server and then forwarded over the wireless link to destination. The actual user message sent in most of the tests was a single character. The average size of the resulting Jabber messages that the server forwards to the recipient client was 160 bytes, comprising the string indicating the source and the overhead due to the XML encoding. The test cases include a situation where the network delays one of the messages sent by the client or the server, and a situation where the fixed client sends messages to the wireless client when it is still disconnected from the Jabber server. Such messages will be delivered from the server to the recipient client, when it reconnects.

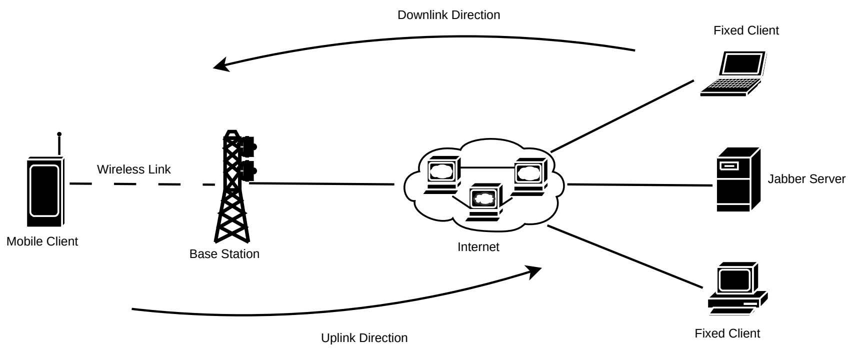


Figure 1. Emulated Network Environment

The wireless link was emulated using the Seawind network emulator [5] developed for emulating wireless networks. For the baseline tests, the wireless link had a bandwidth of 28.8 Kbps and a propagation delay of 200 msec to roughly approximate GPRS-type link characteristics. Test cases were also repeated with different values of link bandwidth and propagation delay. We have chosen a GPRS-type link as the reference as it is an example of a wireless environment where delays and packet losses are likely to occur, but the test runs have general validity.

3. EXPERIMENTAL RESULTS

The experiments showed that the IM server had shortcomings, which affected instant messaging sessions. We discuss the effects of such shortcomings on the delivery of messages to the wireless client and analyze the causes that provoke them. For brevity reasons, we do not go into the details of the test runs, but discuss the main considerations we have drawn after analyzing the results. These considerations will be the basis for the guidelines, which we provide in Section 4.

3.1 The effect of Nagle algorithm on the size of the TCP segments

In various test scenarios several messages were sent continuously from a fixed client to the mobile client. According to the Jabber protocol, the messages are first routed to the server where the source client is registered, and then delivered to the intended recipient, after the proper processing. In our experiments, we noticed that the typical behavior of the server was sending one Jabber message in a TCP segment. This obviously results in inefficient use of the network resources as Jabber messages are usually much smaller than the TCP Maximum Segment Size (MSS) in use. Therefore, the protocol header overhead tends to make up a significant proportion of the bytes delivered over the wireless link.

The reason for this behavior is that the arrival rate of the instant messages at the server for a given recipient is often not higher than the round-trip time (RTT) between the server and the recipient client. When this is the case, the TCP Nagle algorithm [7] is not able to prevent the server from sending small TCP segments. According to the Nagle algorithm, when a TCP acknowledgement for a small TCP segment, that is, smaller than the TCP MSS, is pending, no additional small TCP segments can be sent to the network until the acknowledgement has been received. This would effectively combine several consecutive instant messages into a single full-sized TCP segment provided that enough instant messages for the same recipient would arrive at the server and be delivered to the TCP layer meanwhile the server is waiting the TCP acknowledgement for the previous small segment.

In our tests, no new instant messages typically arrived at the server before the TCP acknowledgement for the TCP segment carrying the previous message had arrived. The Nagle algorithm was effective only in certain special cases, like when one of the messages forwarded by the server gets delayed on the wireless link and does not reach the recipient client “immediately”, causing a delay in receiving the TCP acknowledgement from the mobile client.

Even if the arrival pattern of the instant messages at the server would allow the Nagle algorithm to work, it would still result in suboptimal usage of the network resources as the first of the messages arriving in a row at the server would always be delivered in a TCP segment of its own. Furthermore, the number of messages arriving within a single RTT would often be not enough to fill up a full-sized TCP segment, at least if RTT is rather low. Of course, the server should not wait new messages to arrive for too long either as it would harm the interactivity of the instant messaging.

3.2 Fetching stored messages at the server

In some IM systems, like Jabber, when messages are sent to a disconnected recipient client, IM servers store the messages to deliver them upon the recipient client reconnection. The typical behavior of the server discussed above is to send one Jabber message in a single TCP segment. However, this is not acceptable when the server has plenty of messages ready to deliver to the same recipient client. The ideal behavior of the server is to minimize the number of TCP segments used to deliver the previously stored messages to the recipient by sending full-sized TCP segments.

We emulated the scenario of client reconnection with variable RTT between the server and recipient client and with varying load on the server, in terms of the number of instant messages to handle within a short time period. We observed that when the RTT was short or the server load high the server was sending one small Jabber message in a TCP segment, even though it had several small instant messages ready to be delivered to the recipient. The reason for this behavior was inefficient database access.

Particularly, the server fetched one Jabber message at a time from the database. If the RTT is low, the TCP acknowledgement for the previously sent segment arrived before the server was able to access the next message from the database. Therefore, the Nagle algorithm was not effective in packing several user messages in a single TCP segment, resulting in TCP to transmit each Jabber message in its own TCP segment. Similarly, even if the RTT was relatively long but the load on the server was high packing several messages in a TCP segment was not realized. This kind of misbehavior adds to the protocol overhead and results in inefficient use of the network resources.

This shows that the way of accessing the messages from the database and the database access time plays an important role in the behavior of IM servers. Ideally, before the server delivers any message data to the TCP layer it should fetch from the database as many instant messages as would be needed to fill up a MSS-sized TCP segment. In addition, the access time to the stored messages should be optimized in order to speed up message delivery, Messages can be stored either in the file system or in a dedicated database; the server used in the experiments used Berkeley Database [1] for storing messages and user information. The database access should not constitute a bottleneck, in any network and server load conditions. Low database access times would allow the server to deliver several MSS-sized data blocks quickly to the TCP layer and thereby take advantage of the bulk data transfer capability of TCP.

Cellular networks, the primary target emulated environment, have higher values of RTT than those with which we observed the server misbehavior. Nevertheless, optimizing server access to the database constitutes anyway a relevant performance improvement for any network environment.

4. GUIDELINES FOR INSTANT MESSAGING SERVERS

Ideally, an IM server should always send full-sized TCP segments as long as this does not harm the interactivity of data transfer. In this section we present guidelines for more efficient implementation of IM servers, based on the shortcomings observed in the experiments. The list of guidelines is not and cannot be exhaustive, as many are the aspects where an IM server can be improved; however, it constitutes a starting point and a reference for those who want to implement an efficient IM server.

Guideline 1: Amount of data delivered to the TCP layer

An IM server should deliver stored messages to the TCP layer in large enough data blocks. Each block written to TCP layer should have size of at least one TCP MSS, if possible. The server must minimize the number of RTTs needed to deliver messages to a client.

This guideline should be strictly observed, i.e., in case of delivery of previously stored messages to a newly connected client. The key idea is that the server sends as many messages as possible in a single TCP segment. This approach would reduce the number of TCP segments on the air. Less segments on the air implies lower protocol header overhead. The protocol overhead is not meaningful for the end user, but in cellular networks the user pays for it anyway and it consumes the system resources, so it is desirable to minimize the overhead. The price to pay is a slight decrease in the interactivity of IM sessions.

Guideline 2: Pacing of TCP segments sent to network

IM servers should employ pacing of message delivery to the TCP layer.

Enabling Nagle does not result in efficient message delivery. Therefore, we propose to give the control on pacing the messages to the application, that is, the server. The control is exploited by means of a timer-based mechanism in which the IM server does not deliver Jabber messages to the TCP layer immediately, but waits for the expiration of a timer or that enough Jabber messages have accumulated to send a full-sized TCP segment. The timeout value is tuned so that the number of small segments on the network is limited to at most one per RTT. At the same time, the timer value should be highest possible value that still allows interactive delivery of messages. This allows limiting the number of small TCP segments in the network and at the same time preserving the interactivity of the message delivery, creating a middle ground where the benefits of the Nagle algorithm can coexist with the needs of IM sessions.

Guideline 3: Handling big messages

IM Servers should forward to TCP layer instant messages larger than the TCP MSS as soon as they receive the first MSS bytes of data instead of waiting that the whole message is stored before beginning to forward it.

Such an approach allows for a more timely delivery of the whole message to the recipient. If the server begins to forward the message as soon as it processes the recipient name, when the whole instant message will be received from the source, it is likely that part of it will be already stored at recipient. In wireless networks, which are generally slower than wired ones, this has a particularly positive effect on the performance.

Sending big instant messages is not so common, as the common behavior for the most of the IM users is to quickly send small messages. Nevertheless, situations where users send big instant messages are possible, and servers must be prepared to handle them efficiently; for example, a user could copy and paste a text file and send it as an instant message, rather than send it as separate file off-band. In this case, servers should avoid a store and forward policy and begin to forward a large message as soon as the first TCP segment carrying a part of it is received. We point out that the behavior of the server we used was optimal from this point of view and followed the Guideline 3.

Guidelines 4-5: Efficiency of database access

Guideline 4: *IM servers must access the database where they keep stored message in an efficient way. They must minimize the number of database accesses needed for fetching the messages to be delivered.*

Guideline 5: *IM servers should not fetch from the database a single message at a time for sending but at least MSS bytes data, when possible, to give TCP the possibility to send full-sized segments.*

An optimal server implementation must guarantee an efficient database access, even under heavy load conditions. By efficient access we mean minimizing the number of interactions with the database, trying to fetch at least TCP MSS data at once when possible. This means that if the database allows fetching only one message at a time, the server should fetch several messages in a row until MSS worth of data are available before passing them to TCP. Other optimization ways are possible, but we do not focus on them. For example, the load on servers can be diminished by splitting server components over different machines, or duplicating server components so that they can share the load. However, while co-locating all the modules in the same machine may lead to performance degradation under heavy load conditions, splitting the component over several machines increases the time needed for transferring the message between components for processing purposes.

An additional solution for improving the efficiency of message fetching for IM servers is caching messages temporarily instead of storing them into the database only. This is done when the recipient user is on line but the message cannot be immediately delivered due to the timeout algorithm. Caching up to one MSS bytes of data per recipient is enough. If the cache is associated with the timeout algorithm, in fact, as soon as MSS bytes of messages have accumulated in it, they would be delivered to TCP. If more messages are waiting for that recipient client, they are fetched from the database and moved to the cache after the previous MSS chunk was delivered. This size suggestion for the cache is one option; an alternative would be to tune the portion of cache assigned to each active user according to the number of users on line. When a user goes off-line his portion of cache is freed and can be distributed among other users. However, optimal handling of the cache size is out of scope for this paper and we do not discuss it further.

5. TIMEOUT ALGORITHM

The rationale behind the proposal of a timeout algorithm for instant messaging servers is to find the trade-off that the choice between sending full sized TCP segments and achieving interactivity of message exchanges. In the solution we propose an algorithm that the server application should follow in delivering instant messages to the TCP layer; the messages are delivered only when either there are enough messages to fill up a full-sized TCP segment or when a timer has expired, as further postponing of the delivery would cause loss of interactivity.

5.1 Computing the Algorithm Parameters

The timeout value is set such that at most one small TCP segment is sent on the network in an RTT. This solution preserves the efficiency of data delivery implying less small TCP segments in the network and guarantees the interactivity of IM applications as messages would never wait too long at the server before being delivered.

The server should tune the timer value based on the RTT of the connection between the server and destination client, together with taking into account the interactivity requirements of the message delivery. An ideal solution would be for the server to use the RTT values already computed at TCP layer. When TCP layer information on the RTT is not available, the RTT should be estimated at the application layer. As there are no application level acknowledgements in Jabber, the RTT estimation could occur when a Jabber session is established. In this phase, the server sends a digest authentication challenge and the client a response message. In the following we assume that the server is able to estimate the RTT of the connection with a given destination client in either way.

The RTT to be used for calculating the IM server timer value, IMTimeout, takes into account the estimated RTT value (Estimated RTT):

$$IMTimeout = \gamma \cdot EstimatedRTT$$

The IMTimeout should be higher than the EstimatedRTT, which means $\gamma > 1$ but it should not exceed any value that could hurt the interactivity of instant message delivery. We do not enforce any specific value, but leave it as a subject to further study.

5.2 The Algorithm

The high-level timeout algorithm for a given destination client A is shown in Algorithm 1. When A connects to the server, the RTT of the connection must be computed for the first time. The RTT estimate is continuously updated by the server independently from the execution steps of the algorithm according to the chosen method, application level or TCP based. The server maintains a queue, where the pending messages for each destination client are stored until the IMTimeout expires or until the total amount of data in the queue reaches the threshold of TCP MSS bytes.

There are three events that trigger algorithm execution: arrival of a message for the client, timeout expiration, and client disconnection. When a first message in queue for A is received, if the size of the message is smaller than the MSS, the message is left in the queue and the IMTimeout is started. If another message addressed to A is received, it is stored and the size of the queue is checked. If the queue size is under the MSS threshold, nothing happens and the IMTimeout timer continues to elapse.

Otherwise, if the first message was larger than MSS bytes, or if, due to a message arrival the queue exceeds the threshold, the while loop is entered. TCP MSS-sized blocks of data are delivered to TCP until the queue size goes below the threshold. If there are still data in the queue after the delivery of the blocks, they remain stored and the IMTimeout value is recomputed and the timer is started again. If the queue is empty after the last delivery of data, the timer is cancelled. If the IMTimeout timer expires, the pending data are delivered to TCP, and the timer is cancelled.

If the server receives a disconnection request for A, while there are pending messages, it leaves the messages in the database for delivery upon next reconnection of client A. After that, it can execute the disconnection operations, like communicating the change in the presence state of A to the clients that are on the contact list of client A.

6. CONCLUSIONS AND FUTURE WORK

This paper discussed the result of an experimental study about the behavior of an IM server in a wireless environment. We envisage that the use of IM services in such an environment will gain diffusion in the near future. We observed shortcoming in the behavior of the server in and presented guidelines for a more efficient IM server implementation over wireless links. The guidelines are valid in any network scenario, even though the improvement they bring about is particularly effective in wireless networks with scarce resources.

An application-level timeout algorithm was presented, useful for limiting the number of small TCP segments in transit on the network. This algorithm solves the trade-off deriving in IM servers from employing the Nagle algorithm to prevent from sending small TCP segments into the network as the Nagle algorithm only provides suboptimal performance.

The timeout algorithm creates a middle ground by allowing the server TCP layer to always send full-sized TCP segments if enough data is available by delivering large enough data blocks to the TCP layer at a time. The algorithm proposed is still a high level one; future work comprises an implementation of the algorithm, at least in a simplified version of an IM server to tune the γ parameter that defines the length of the IMTimeout.

This would allow a quantitative evaluation of the algorithm to test its effectiveness in meeting the design goals.

Algorithm 1. The IMTimeout algorithm

```
Arrival of a message for client A

Enqueue the message;

if ((message is first in the queue) && (queue  size < TCP MSS))
{
    store it, compute the IMTimeout value;
    start the IMTimeout timer;
}
delivered_flag = FALSE;
while (queue size >= TCP MSS)
{
       deliver MSS-sized chunk of data;
        delivered_flag = TRUE;
}
if ( delivered_flag == TRUE)
{
      if (there are pending data in queue)
         recompute IMTimeout value, restart IMTimeout timer;
      else
         cancel IMTimeout timer;
}
IMTimeout Expiration

Deliver all pending data in the queue to TCP layer;
Cancel IMTimeout timer;

Client A disconnection request

Leave the pending messages in the database;
Execute disconnection operations;
```

Another possible improvement direction is a more refined formula for computing the IMTimeout. For example, its duration can take into account the pace of arrival of messages for a given destination client, so that the length is stretched if the message arrival rate to a destination client is high and vice versa. The rationale behind this proposal is that it is not worth to wait long if messages for that client arrive only once in a while. The guidelines are independent from the TCP version in use; an IM server can effectively apply these guidelines regardless of any TCP enhancements used. TCP increased initial window may contribute to improved message delivery efficiency.

An optimal implementation of an IM server would combine the use of the IMTimeout with the caching of instant messages. Further study on how to dynamically adapt the cache portion based on the number of users on line is required.

REFERENCES

[1] Berkeley DB Database. http://www.sleepycat.com 27-4-2005
[2] Jabber Software Foundation. http://www.jabber.org. 27-4-2005
[3] Jabber Studio Project. http://jabberd.jabberstudio.org/ 27-4-2005

[4] GAIM Multiprotocol IM Client. http://gaim.sourceforge.net/] 27-4-2005

[5] Kojo, M., et al., September 2001. Seawind: A Wireless Network Emulator. In Proceedings of 11th GI/ITG Conference on Measuring, Modeling and Evaluation of Computer and Communication Systems (MMB 2001), RWTH Aachen, Germany, Published by VDE Verlag

[6] Parviainen, P. and Parnes P. March 2003. Mobile Instant Messaging. In Proceedings of the 10th International Conference on Telecommunications, ICT 2003. Volume 1, 23 Feb.-1 March 2003Pp:425 - 430

[7] Nagle, J., January 1984. Congestion Control in IP/TCP Internetworks. RFC 896, Internet Society.

[8] Saint-Andre, P. ed., October 2004, Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920, Internet Society.

[9] Saint-Andre, P. ed., October 2004, Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921, Internet Society.

ENHANCING MOBILE PEER-TO-PEER ENVIRONMENT WITH NEIGHBORHOOD INFORMATION

Arto Hämäläinen and Jari Porras
Lappeenranta University of Technology
Laboratory of Communications Engineering
P.O. Box 20
53851 Lappeenranta
Finland
{Arto.Hamalainen, Jari.Porras}@lut.fi

ABSTRACT

In this paper some enhancements to the performance of mobile peer-to-peer environment are presented. The work is based on our implementation of mobile peer-to-peer communication environment, PeerHood. PeerHood offers direct communication between mobile devices by using Bluetooth, WLAN or GPRS communication technologies. PeerHood is enhanced by allowing devices to exchange their neighborhood information thus extracting better view of the environment. This information can both bypass problems caused by incomplete device discovery and provide information about devices outside the immediate vicinity.

KEYWORDS

Personal trusted device, peer-to-peer, neighborhood, ad hoc networking, device discovery, service discovery

1. INTRODUCTION

Current trends in networking consist of wireless environments and mobile devices. These devices are used both in infrastructure-based and ad hoc networks. Another widely discussed topic is the ubiquitous computing. Until recently, all these concepts have been just ideas because non-existent devices and network technologies. However, recently the utilization of wireless environments like 802.11 wireless LANs and Bluetooth wireless technology has rapidly increased.

Device and especially service discovery is fundamental part for mobile ad hoc environment. Usually presumed roaming and mobility prevent the use of static directory-based service and device discovery and suggest that dynamic service discovery technology could be more suitable. Generally these dynamic service discovery protocols are used to advertise services on own device and search services on remote devices. Each device should have a client part which searches for services on other devices and a server part, which responds service discovery requests originating from other devices [1].

Several service discovery protocols have been developed, but mainly for infrastructure-based networks. These protocols have originated from and developed by both standardization organizations and industrial consortiums. Internet engineering task force has developed Service Location Protocol (SLP), which provides a scalable framework for the service discovery [2]. SLP is intended for networks which have cooperative administrative control. Therefore its usability for ad hoc networking must be inspected before adoption. Sun's Jini, Microsoft's Universal Plug-and-Play and Bluetooth Service Discovery Protocol are examples of industry-originated service discovery protocols. Bluetooth SDP is the only one of these which is targeted directly to mobile wireless network. [1]

Communications in ad hoc wireless networks are often established in a peer-to-peer manner. Peer-to-peer environment implemented in mobile devices assists importing applications to mobile ad hoc networking. In this paper some enhancements to the basic operation of our mobile peer-to-peer environment, PeerHood, are presented. First the PeerHood environment and its structure as well as operation is explained. Then the neighborhood information exchange and ways to utilize it are examined. Finally some scenarios and results for the evaluation of the enhanced PeerHood are given.

2. MOBILE PEER-TO-PEER ENVIRONMENT

The development in mobile devices, communication technologies and networking paradigms during the last decade has been rapid. Computational power of the mobile devices has increased, new short range networking technologies has emerged and personal networking paradigm has evolved. Although each of the developments alone is significant the combined effect is even more remarkable. Personal information management as well as intelligent and possibly seamless connectivity is possible if considering a powerful mobile device capable of short range communications. By allowing peer-to-peer connections between devices (without network infrastructure) new possibilities will emerge. Currently peer-to-peer approach is mainly used in fixed networks but some approaches to apply it to mobile networks have been proposed. Peer-to-peer communication can be seen as one of the most promising communication paradigms of the future as it allows resource sharing in a flexible manner. It allows truly dynamic networking and is suitable for mobile networks as well. There exist several approaches for the personal networking ranging from proximity based approaches, e.g. Digital Aura [3] and Virtual device [4], to approaches based on personal interests, e.g. Personal Networking [5], I-centric communications [6] and Personal Distributed Environment [7]. We propose another approach, namely peer-to-peer neighborhood, PeerHood. PeerHood is an implementation of a personal area network (PAN) based on peer-to-peer paradigm in mobile environment [8]. PeerHood is built inside a mobile device, i.e. Personal Trusted Device (PTD). Our approach is based on the need of local services (proximity) and the use of them through different networking technologies (Bluetooth, WLAN and GPRS). **Figure 1** presents the idea of PeerHood. Our PTD, i.e. mobile phone in this case, detects other devices in its vicinity. The size of the vicinity, i.e. neighborhood, depends on the communication technology used for observing. **Figure 2** presents another case where the neighborhoods of devices are not of the same size. It should be noticed that some of the devices might not be visible for all the other devices.

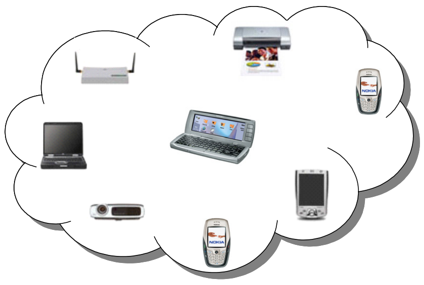


Figure 1. PTD and its neighborhood.

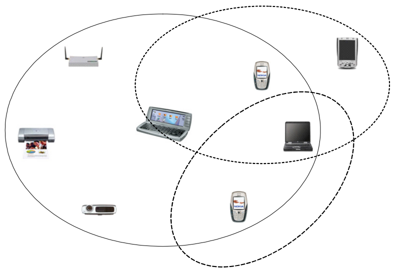


Figure 2. Neighborhoods of different devices.

The main goal of our PeerHood system is to provide a communication environment where devices act and communicate in a peer-to-peer manner. This means that devices communicate directly with each other without any centralized servers, i.e. PTD might communicate with all the devices in its neighborhood. In order to enable fast creation of the required ad-hoc type networks the immediate neighbors of a device are monitored and the gathered information is stored for possible future usage. This is presented in **Figure 3**. Information concerning devices or services within the neighborhood is stored into appropriate place in the PeerHood system. The second goal is to create a library that enables the usage of any supported networking technology via a unified interface so that the underlying networking structure is hidden from the applications point of view. As a direct consequence the application development time should be reduced because complex tasks like device discovery, connection establishment and error checking are handled by the PeerHood system. This library is shown as an interface to the environment managed by the PeerHood layer.

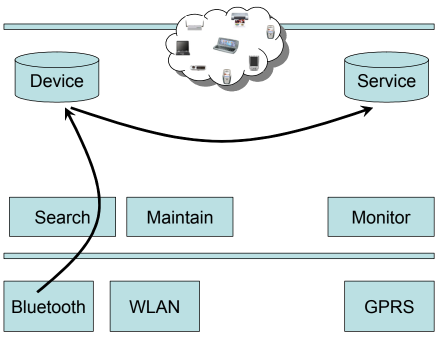
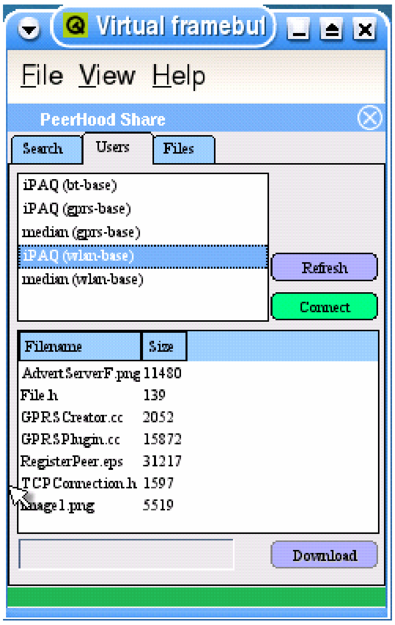


Figure 3. PeerHood layer in PTD.

In our approach the PTD is continuously sensing its neighborhood through different network technologies and it maintains the gathered information for the further usage. As other devices are observed they are added into the neighborhood of the PTD. The seamless connectivity is provided by the PeerHood layer through some basic operations i.e. search, monitor and maintain operations of the changing ubiquitous environment. As new devices and services are observed they are stored into neighborhood information tables. Thus the PTD always has up-to-date information concerning its environment. This information is provided for the application through the PeerHood interface. With PeerHood interface applications can list devices and services in their wireless neighborhood and connect to them when necessary. **Figure 4** presents an example of a file sharing application on PeerHood. This application shows the neighborhood (devices in upper box with their interfaces) and files within the selected device.



iPAQ (bt-base)
iPAQ (gprs-base)
median (gprs-base)
iPAQ (wlan-base)
median (wlan-base)

Filename	Size
AdvertServerF.png	11480
File.h	139
GPRSCreator.cc	2052
GPRSPlugin.cc	15872
RegisterPeer.eps	31217
TCPConnection.h	1597
Image1.png	5519

Figure 4. Mobile file sharing over PeerHood.

3. PEERHOOD IMPLEMENTATION

PeerHood operations have been implemented both in Linux based PDA and Symbian based mobile phone. In this paper we concentrate on the Linux implementation and its enhancements.

Linux implementation of the PeerHood environment is based on four elements, *PeerHood Daemon, PeerHood Library, Additional Components* and *PeerHood Applications*. Of these components the daemon and the library are mandatory for any application to work and thus will be presented in this paper. Additional components are optional middleware libraries that further extend the PeerHood's functionality and are presented in other papers [9]. **Figure 5** presents the components of the Linux based implementation of the PeerHood. The daemon is one independent process which takes care of locating other devices. The daemon implements device discovery using network specific *plugins* through the plugin-interface. Bluetooth, WLAN and GPRS plugins are available for the Linux PeerHood. The library interface provides all the PeerHood functionality to the application.

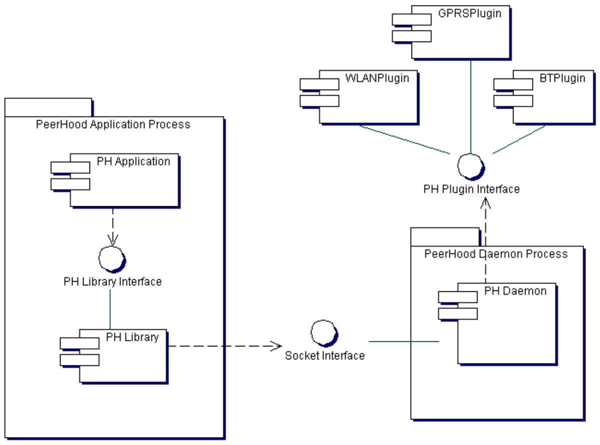


Figure 5. Components of the Linux PeerHood

Because the daemon and the library are independent components, PeerHood class structure is also divided into two parts. Important part of whole PeerHood structure is the abstraction between networking specific classes and interfaces. PeerHood allows developers to add new plugins which can be used by the daemon. To be able to provide this architecture, the whole technology-specific functionality must be contained in one class which implements the specific interface. This allows the core components to remain the same while adding new functionality using plugins. Daemon class diagram is illustrated in Figure 6.

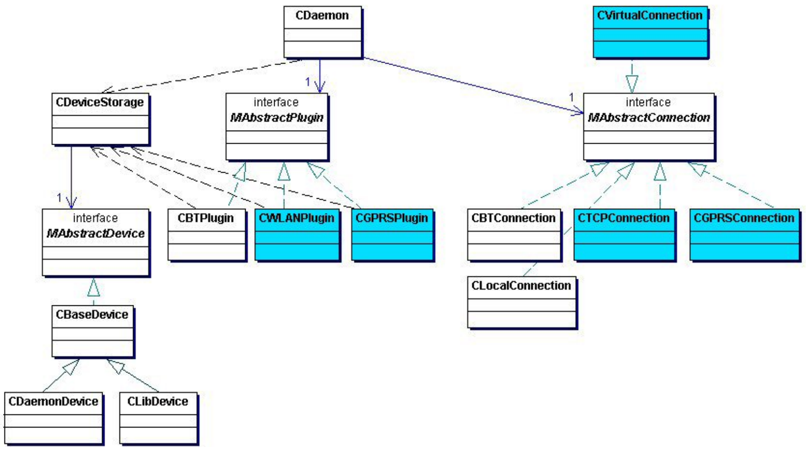


Figure 6 - PeerHood daemon class diagram

The main issues towards the goals of this paper are in the management of the neighborhood. This means that the functions taking care of finding and monitoring devices in the environment need to be studied. To find and keep track of devices in the neighborhood, PeerHood uses different device discovery functions of different network technologies. While using Bluetooth connections PeerHood can utilize its service discovery protocol, for WLAN or GPRS some specific methods has been implemented for discovering devices. These device discovery functions are not always the most efficient methods, i.e. the frequency of inquiries might degrade the performance of the whole system. In this paper we have focused to handle this problem with the neighborhood information exchange approach. By adding some neighborhood information into the inquiries the performance of neighborhood information management is improved.

4. NEIGHBORHOOD INFORMATION EXCHANGE

Mobile ad hoc and infrastructure-based networks have different requirements and facilities for service discovery. Peer-to-peer mobile networks are generally formed in an ad hoc manner. However, some of the devices of an ad hoc network can also be connected to an infrastructure-based network or their environment in an ad hoc network is changing slowly. Therefore they likely have more comprehensive information about the devices around than passing mobile devices. For this kind of situations where mobile ad hoc networks meet networks based on fixed infrastructure, exchange of neighborhood information may turn out to be valuable element of service discovery.

Neighborhood information exchange has been integrated to PeerHood operation. Neighborhood information is gathered from other PeerHood devices and it's used together with regular device discovery to provide constant view of devices nearby and further away. A message sequence chart for neighborhood information exchange can be seen in **Figure 7**.

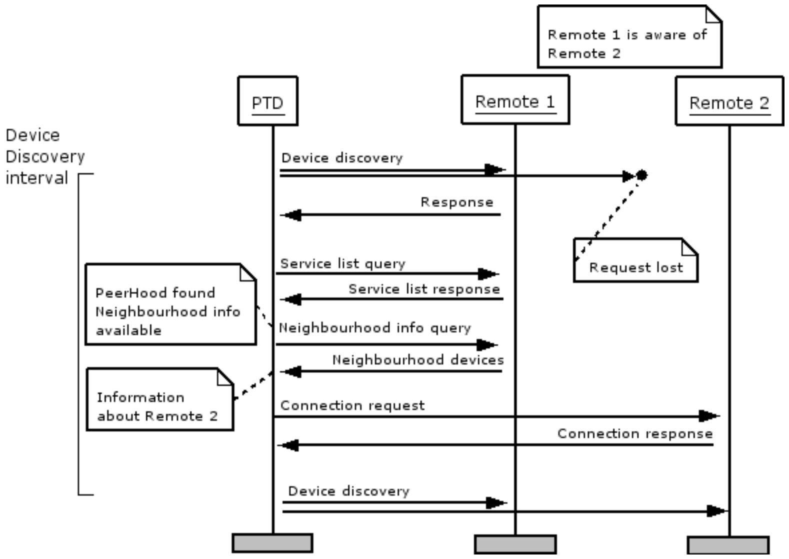


Figure 7 - Message sequence chart for neighborhood information exchange

In this example, *Remote 1* provides neighborhood information exchange service to other devices. Mobile Personal Trusted Device *PTD* is sensing its neighborhood with regular *Device discoveries*, which are performed at certain interval. Without neighborhood information exchange, failure to discover a device offering a desired service, in this case *Remote 2*, could postpone connection to service until the next device discovery. By using neighborhood information exchange, however, *Remote 1* provides the requesting *PTD* all the needed information about the *Remote 2.*

Neighborhood discovery is carried out by the PeerHood daemon class, whose structure is presented in Figure 5. Daemon is the component in the PeerHood, which takes care of locating other devices and provides applications with an interface to connections. Daemon also acts as a service discovery server, advertising services belonging to PeerHood applications running on a device. It uses different plug-ins (Bluetooth plug-in, WLAN plug-in and GPRS plug-in) through the PeerHood plug-in interface. The clients can connect to a daemon and send requests to it through a socket interface provided by the PeerHood library. Commands include listing of found devices and services, registering a new service and monitoring a connection. Although neighborhood information inquiry in general is a common operation for all network technologies, it is implemented within plug-ins alongside co-existing device discovery functions. Discovering available PeerHood services or neighboring PeerHood devices from another PeerHood device can be done consecutively or separately.

If a device gathers information about neighboring devices and wants to provide it to other devices also, the flag for neighborhood information exchange service is registered to PeerHood. When a device finds the neighborhood information flag on a remote device using service discovery, it requests the neighborhood information from it. The remote device then compiles the list of neighboring devices and services and sends the list to the requesting device.

Device information received by neighborhood information inquiry is stored to the device storage (CDeviceStorage) just like information received by regular device discovery methods. Along with regular information, address of device, where the neighborhood information was received, is stored. This way the connection to a service can be established through the intermediate device, if possible.

5. USAGE SCENARIOS

A few usage scenarios are designed for the neighborhood information. In a general level in these scenarios neighborhood information is used both to provide more comprehensive knowledge of the neighborhood and to gather information about devices outside the immediate neighborhood.

5.1. Personal networking outside the immediate vicinity

Personal networks are formed around a person and they consist of devices which are most likely to be used by him in his everyday routines [5]. Personal network is not limited to nearby devices within the reach of wireless short-range technologies, but may be extended via Internet or multi-hop wireless network to another set of devices. The use of neighborhood information provides information about distant devices. If an interesting device or service is found using neighborhood information exchange, then a connection could be established either directly using long-range network technology or via one or several intermediate devices, which may be the same devices that forwarded the device information in the first place.

5.2. Exchanging detailed device information

Advanced mobile devices could be used in several purposes in addition to regular personal communications and information management. The processing power and storage capacity is increasing rapidly and their support for different communication is becoming more and more comprehensive. These factors combined with increasing popularity of these advanced mobile devices mean, that there's quite a lot processing power available around us in everyday wireless neighborhood. Surrounding devices could be used in parallel manner to solve large computational problems.

In addition to authorization and compatibility issues, the ability to survey available resources is important element in this kind of operation. Resources can be scanned by exchanging detailed device information between mobile devices. This information includes information about the processing power, storage capacity and available services available at each mobile device.

5.3. Dedicated service discovery

Another usage scenario is based on presence of certain devices in certain locations. For example at home or at work there’s a personal computer available practically always. Instead of running device discovery constantly itself, a mobile device may use a dedicated computer to do the task. When information about other devices is needed, the mobile device can connect to the PC and request the information. An advantage is gained by this method, if the PC has other network connectivity not found in the mobile device, is that the mobile device gets information about devices which are not reachable by its own discoveries. If an important service is found on a device for instance in another floor or department nearby, the mobile device user may connect to the service using the PC or walk closer to the service and use it with his mobile device. If the mobile device has the needed connectivity available but disabled due to power consumption or any other reason, it may be enabled and the found service connected.

Another advantage of this method is the avoidance of battery consuming inquiry process in the mobile device. Determining the power consumption of Bluetooth or other wireless devices is difficult if the exact mode of operation is not known. While constant transmitting and receiving could lead to an early drain of batteries, idle or sleep modes will make it last a lot longer [10]. Although an idle Bluetooth chip requires in active mode almost half of the current compared to the send or receive modes operation (50 mA), the use of low-power modes reduces the power consumption drastically (down to 60µA) [11]. While device discovery modes differ from technology to technology, their operating scheme is similar. A device transmits a request to the network and begins listening for responses. When this is done constantly, the utilization of low-power modes is not possible.

6. EVALUATION

Evaluation was done in a Bluetooth environment similar to one that is pictured in Figure 7. The objective of the tests is to indicate enhancement in PeerHood operation by introducing neighborhood information exchange. The measurements are shown in the Figure 8. Ten tests were carried out and measured for both with and without neighborhood information exchange. Time measured was the time between the launch the PeerHood daemon in PTD and discovery of two other PeerHood devices located in a same room. Other Bluetooth devices were also detected, but the detection of these two PeerHood devices was considered the criteria for a completed test.

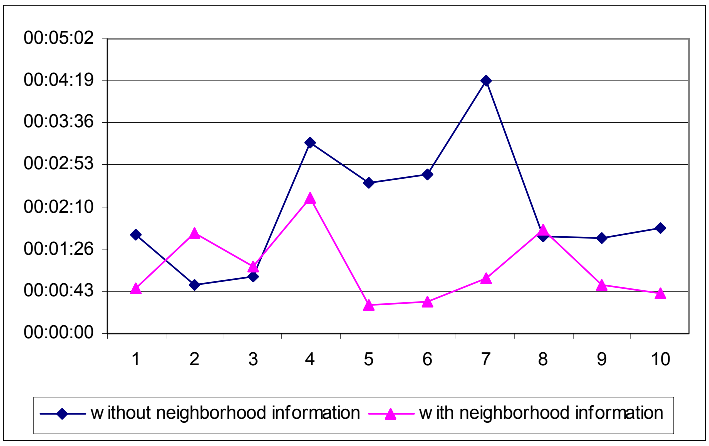


Figure 8 - Time taken to find two PeerHood devices with and without neighborhood information exchange

One of the other devices was a stationary device which was offering neighborhood information and had a 30 second interval between its own device discoveries. The other was acting as a mobile device with an interval of 10 seconds between device discoveries. This reflects the heterogeneity of devices in a real environment. The device and service discovery on the mobile device was often interfering our device discovery and the device remained undiscovered. However, the stationary device with a longer device discovery interval was found more certainly. As soon as it was found, the neighborhood information could be inquired and exchanged, and the mobile device was also discovered. The average time taken by the discovery was 77 seconds with and 129 seconds without neighborhood information, so these results show that neighborhood information enhance the operation in this kind of environment. Furthermore, implementing advanced fixed and mobile device roles could improve the operation even more.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a mobile peer-to-peer based environment, PeerHood, and proposed some enhancements to its operation. These enhancements allow us a faster gathering of the neighborhood information and thus improve the whole environment.

Exchanging information like device addresses, names and service descriptions causes also worries about the security of the system. A suitable solution could be implementation of a generalized authentication module, which is used by other services and applications. This way, authentication policies for each service could be managed in a one module.

REFERENCES

[1] Golden G. Richard III: Service and Device Discovery: Protocols and Programming, McGraw-Hill, 2002.

[2] Guttman E., et al.: Service Location Protocol, Version 2, IETF Request for Comments 2608, June 1999. Available at: http://www.ietf.org/rfc/rfc2608.txt

[3] Ferscha A, et al.: Digital Aura, Advances in Pervasive Computing, part of the Second International Conference on Pervasive Computing (Pervasive 2004), Austrian Computer Society (OCG), Vol. 176, pages: pp. 405-410, April 2004

[4] Jonvik Tore E., et al.: Building a Virtual Device on Personal Area Network, Proceedings of 2003 International Conference on Software, Telecommunications and Computer Networks (SoftCOM'2003), Dubrovnic (Croatia) / Ancona, Venice (Italy), October 7-10, 2003.

[5] Niemegeers I. and Hememstra de Groot S.: Personal Distributed Environments for Mobile Users, Mobile Europe 2003, European conference on mobile solutions, 2003. Available at http://katanga.bbn.de/mobile_europe_2003/

[6] Arbanowski, S., et al.: I-centric communications: personalization, ambient awareness, and adaptability for future mobile services, Communications Magazine, IEEE, Volume: 42 , Issue: 9 , Sept. 2004.

[7] Atkinson, R. C., et al.: The Personal Distributed Environment, Wireless Personal Multimedia Communications 2004, September 2004.

[8] Porras, J., et al.: Peer-to-peer communication approach for mobile environment, 37th IEEE Annual Hawaii International Conference on System Sciences (HICSS), 2004.

[9] Jäppinen, P., et al.: ME: Mobile E-Personality, WSEAS Transactions on Computers, Volume 2, Issue 2, April 2003.

[10] Morrow, R.: Bluetooth operation and use, McGraw-Hill 2002.

[11] Zhang X. et al.: Bluetooth Simulations for Wireless Sensor Networks Using GTNetS, The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04), 2004.

A GATEWAY FOR SIP EVENT INTERWORKING

Sasu Tarkoma and Thalainayar Balasubramanian Ramya
Helsinki Institute for Information Technology
P.O.Box 9800,
FIN-02015 HUT, Finland
{sasu.tarkoma, ramya}@hiit.fi

ABSTRACT

Session Initiation Protocol (SIP) is an application-layer control protocol for creating, modifying and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. SIP includes an extensible framework for asynchronous event notification. This paper investigates the interworking of SIP events with a generic publish/subscribe system. Publish/subscribe systems typically support anonymous one-to-many form of communication, which is not supported by the default SIP event package. We present the design and implementation of a gateway component, and discuss supporting expressive subscription semantics using filters and also mobile clients.

KEYWORDS

SIP, Gateway, Publish/Subscribe, Interworking

1. INTRODUCTION

The Session Initiation Protocol (SIP) is an application-layer protocol for establishing and controlling sessions (Rosenberg 2002). SIP has been extended with support for asynchronous event notification. In this paper, we discuss our experiences with bridging the SIP event domain with a generic publish/subscribe (pub/sub) event API. The models differ, because the SIP notification model does not support anonymous one-to-many event dissemination with filters. Interworking is realized using a stateful gateway component, which is based on the bridge pattern (OMG 2004).

Distributed publish/subscribe systems consist of three entities: subscribers, producers, and the event service. The event service is a logically centralized component that provides interest registration services for subscribers and API calls for producing events (Eugster 2003, OMG 2001). The service ensures that published events are delivered to proper subscribers that have previously expressed interest in receiving events. The event service decouples subscribers and publishers of information, and also allows anonymous information delivery. Expressive, content-based pub/sub systems have been proposed for the development of mobile applications, because they support run-time reconfiguration and adaptation.

The SIP event model supports one-to-one and one-to-many communication between SIP entities. The SIP event model requires that subscribers register directly with an event producer. The event model does not specify a logically centralized component for connecting subscribers and publishers. The model is simple and flexible, but lacks the features of distributed pub/sub systems, such as expressiveness and anonymous communication. The differences between the SIP model and the generic pub/sub model complicate interworking of SIP event domains and other pub/sub domains. Interworking support is motivated by the benefits of the generic pub/sub model and universal data access.

This paper is structured as follows: Section 2 presents the motivation and Section 3 the SIP event framework. Section 4 discusses the generic event model in more detail. Section 5 presents the gateway design and implementation. Finally, Section 6 presents the conclusions.

2. MOTIVATION

A content-based pub/sub network facilitates the integration of systems, such as active badge systems, proximity sensors, information and multimedia delivery services, and positioning systems. Typical SIP applications are call control, presence management, and instant messaging (IM). SIP has also been proposed for ubiquitous and context-aware computing (Berger 2003) by using the presence extensions (IETF 2005) and the event package (Roach 2002). By seamlessly integrating and combining information from both SIP and pub/sub domains, we can create more expressive application behaviour, such as location-based buddy lists. In addition, by using server-side filtering part of the application complexity can be pushed into the infrastructure.

For example, consider the case of a WLAN-based positioning system in a hospital. This enables the tracking of doctors, nurses, patients, and medical equipment within the building. A doctor can subscribe tracking information to his SIP-based mobile phone or PDA and receive updates using wireless communication. A content-based routing network may be used to route information between various entities. The motivation for using the pub/sub network is its flexibility and expressiveness in delivering information.

The gateway connects the SIP domain to the pub/sub network. Hence, the gateway also brings the benefits of filtering to SIP clients and pushes this complexity to the edge of the SIP domain. Filtering can be made transparent to existing SIP applications by using web-based technologies. A conventional SIP buddy list and instant messaging application may be used if the filters at the gateway are configured using a web browser. The notifications routed from the pub/sub domain based on a set of filters are transformed into SIP notifications at the gateway and they may be directly used by SIP-aware applications, such as buddy lists or IM software. For example: the buddy list of a Doctor could be updated based on people's location without changing the semantics of the buddy list application. This would require transformation rules being set at the gateway for the SIP notifications to be compatible with the buddy list application. Another example is pushing interesting multimedia content to end-user consumers. In this case, subscribers subscribe content and specify the filters in the application or they preconfigure the preferences at the gateway (or some other server).

The gateway enables and supports applications that require expressive notification. The pub/sub network could be thought of as a flow of rich information that is delivered based on expressive filters. The SIP domain, on the other hand, can be thought of as an edge domain for this content that connects various wireless and mobile devices to this information pool. Context-aware applications are an important application area for the gateway, for example the location-based notification of the hospital scenario. The system should be able to realize interactions such as "shutdown all robots in room 100" or "Alice has left room 100". The gateway may also provide more flexible support for disconnected operation by buffering incoming notifications and support various queuing policies. One way to realize context-based addressing of an entity is to use content-based routing and subscribe the current context of the entity.

3. SIP EVENT FRAMEWORK

The SIP event package enables a client to subscribe to the desired events and receive notifications when the expected event occurs (Roach 2002). The main application areas for the SIP event package are callback services, buddy lists (presence), and message waiting indications. The SIP domain consists of several entities. We concentrate on the two most important components: the SIP client and server. The SIP client is a network element that sends SIP requests and receives SIP responses. Clients interact directly with a human user or a terminal device. Proxies and user agents are clients. The SIP server is a network element that receives requests and sends back responses to requests. Servers include proxies, user agent servers, redirect servers, and registrars.

The SIP event framework supports two methods for event subscription and notification, namely, SUBSCRIBE and NOTIFY. The SUBSCRIBE method is used to request the current state and state updates from a remote entity. The NOTIFY method is used to notify the subscriber when the requested event occurs. Figure 1 shows the sequence diagram of sending a subscribe request and receiving a notify response.

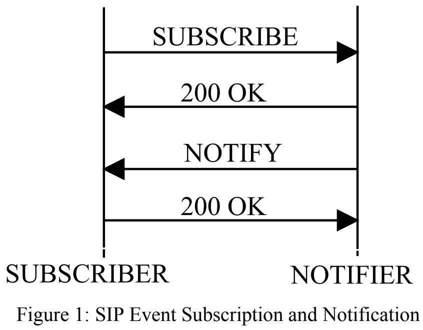


Figure 1: SIP Event Subscription and Notification

The headers of the SUBSCRIBE and NOTIFY messages include values for expiry time, request URI, and an event header. A Call Sequence value is stored in the header to maintain the order of the transaction, while a Call Identifier value uniquely identifies the particular invitation or registration from the client. Figure 2 presents the fields of the SUBSCRIBE and NOTIFY headers. A subscribe header message with the expiry field value set to zero denotes an unsubscribe request.

```
SUBSCRIBE sip:notifier address SIP/2.0
Via SIP/2.0/UDP origin
Max-Forwards:hop count
To: <sip:notifier address>
From: <sip:subscriber address>; tag=122
Call-ID: call-id value
Cseq: call sequence number
Allow-Events: supported event packages
Contact:<sip:subscriber contact>
Event: package name
Content Length: 0
```

```
NOTIFY sip:subscriber address SIP/2.0
Via SIP/2.0/UDP origin
Max-Forwards:hop count
To: <sip:subscriber address>
From: <sip:notifier address>; tag=122
Call-ID: call-id value
Cseq: call sequence number
Contact:<sip:subscriber contact>
Event: package name
Subscription-State: subscription status
Allow-Events: supported packages
Content-Type: message type
Content Length: 0
```

Figure 2: SIP subscription and notification headers

The SIP framework supports four different types of mobility (Schulzrinne 2000): *session mobility* allows a user to maintain a media session while changing terminals, *terminal mobility* allows a device to move between IP subnets while continuing to be reachable for incoming requests and maintaining sessions across subnet changes, *personal mobility* allows the addressing of a single user located at different terminals by using the same logical address, and *service mobility* allows users to maintain access to services while moving or changing devices and network service providers. These different types of mobility support apply also for event delivery and subscribers receive notifications even when they are roaming. Mobility support is realized by updating any client address changes to respective servers. SIP supports personal mobility by using the forking technique (Rosenberg 2002).

4. EVENT SYSTEMS

Event-based systems are seen as good candidates for supporting distributed applications in dynamic and ubiquitous environments because they support decoupled, anonymous, and asynchronous one-to-many information dissemination. Event systems are widely used, because asynchronous messaging provides a flexible alternative to RPC (Remote Procedure Call) (Colouris 1994). In the general model of event notification, subscribers subscribe events by specifying their interests using filters. Filtering is central core functionality for realizing event-based systems and accurate content-delivery. The main motivation for filtering is to improve accuracy in information delivery by delivering only those messages that are interesting for a client of the system — the delivered messages must match the filters defined by the client. Filters and their properties are useful for many different operations, such as matching, optimizing routing, load balancing, and access control.

Typically, the event service interface consists of the basic primitives: *subscribe(F)*, *unsubscribe(F)*, and *publish(N)*. Figure 3 shows the basic operations supported by the generic publish/subscribe event model. *F* denotes a filter and *N* a notification. The filter is a stateless Boolean function that takes a notification as an argument. The notification is usually represented as a list of typed tuples, but it may also be structured, for example an XML fragment. The API may also be extended with advertisements.

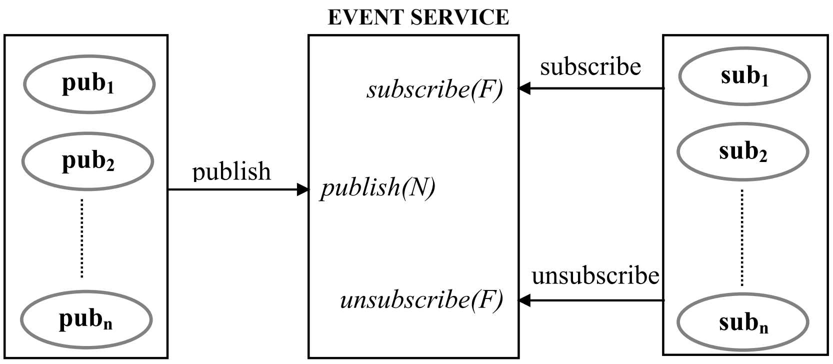


Figure 3: Generic publish/subscribe event model

For the gateway design presented in this paper, we leave the exact nature of the filters and notifications open. The pub/sub event domain part of the gateway implementation was implemented using the Fuego event system (Tarkoma 2003, 2005). The Fuego event system consists of a number of event routers connected into a routing topology. The system divides a router into two parts: the access server part and the routing core part. The former is responsible for event delivery and buffering for local clients, and the latter is responsible for distributed event routing. The routing core is based on a set of efficient data structures for content-based routing. Supported configurations include hierarchical, event channel, and peer-to-peer routing.

5. SIP EVENT GATEWAY

The aim of the gateway design is the interworking of two different and non-interoperable event systems, namely the SIP event domain and the generic pub/sub domain. The gateway should be efficient, impose no modifications to the event APIs of the internetworked domains, and also be transparent for applications and servers. In this section of the paper, we present the gateway design and implementation. We assume that both event domains have unique identifiers for subscriptions and that each event has an explicit or implicit type. We also assume that the SIP client is able to send subscription and notification messages and that the clients know the name of the gateway component.

5.1 Design

Figure 4 presents an overview of the gateway design with messaging abstracted using input (I) and output (O) interfaces. Both interfaces must support subscription, unsubscription, and notification (publishing events). The gateway uses the event API of the two domains.

Therefore, it is a client of both domains and does not require any changes to the way the domains operate. The gateway is a stateful entity and it tracks active subscriptions for both domains.

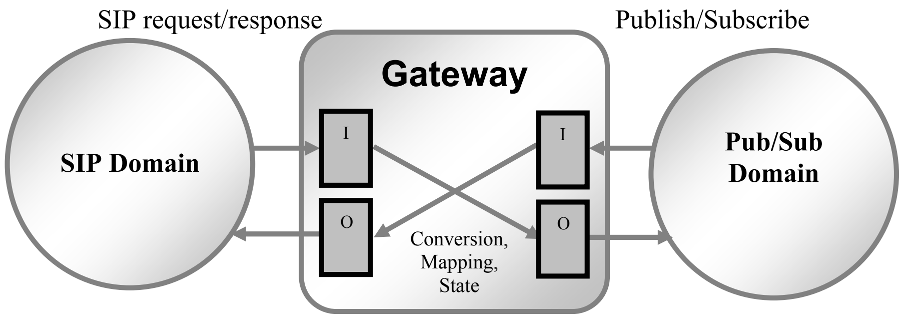


Figure 4: Gateway design

SIP clients must register explicitly with the gateway in order to receive events from the pub/sub domain. In our design, the SIP proxy handles client registrations. The proxy maintains a list of registered domains and routes the SIP requests/responses to the gateway that are intended for the pub/sub domain. The registration may include a filter and the type of the filter. The gateway only accepts registrations with the filter types it supports.

One of the central challenges of the design is how SIP event producers are informed of active subscribers. If filters are used, the producers have to explicitly register at the gateway in order to receive a subscription request, which then may result in NOTIFY messages being produced. If the SIP event producer does not support source-side filtering, the gateway is responsible for filtering. The producers may be preconfigured to avoid changes to applications. If entity-to-entity event delivery is used based on the name of a SIP entity, the subscription request may be forwarded without prior registration at the gateway. The subscription request to SIP event producers is needed to avoid unsolicited messaging.

The gateway is also a client to the pub/sub system. The gateway API is exposed using the generic pub/sub system by subscribing an event-type / attribute with the gateway name. This is the gateway service subscription. Any subscribers in the pub/sub domain need first to subscribe the event they are interested in and then to contact the gateway. A subscriber contacts the gateway by simply publishing an event that matches the gateway's service subscription, and including in the event any relevant subscription information. Unsubscription is similar to subscription, but instead of creating state at the gateway and possibly at SIP producers, it removes state. Publication of events is transparent for the pub/sub domain, because the gateway uses the subscribe() - primitive in the client API. A similar pub/sub API-based mechanism is used in the Siena mobility support service (Caporuscio 2003) to support mobile subscribers.

All subscribe and unsubscribe operations are acknowledged by the gateway when it receives a reply from the server it is using through the pub/sub API. Acknowledgement semantics differ in the two domains and in the pub/sub domain notifications are not necessarily delivered until the subscription message has propagated throughout the event network.

The proposed gateway mechanism supports scalability by partitioning event types into different gateways. Different event types are independent and thus may be handled by different gateways. This kind of partitioning works for both SIP and pub/sub domains. In the SIP domain, the gateway can be selected at lookup time. In pub/sub domains, each gateway subscribes the event types it supports. Therefore, having multiple gateways does not cause inconsistency problems for content-based routing.

5.2 Main Functions

The four main functions of the gateway are as follows: 1. to receive SIP messages (requests/responses), 2. convert and forward SIP messages to the pub/sub domain, 3. receive pub/sub messages, and 4. convert and forward pub/sub messages to the SIP domain. Figure 5 illustrates the different steps in SIP subscription and Figure 6 shows the steps needed for performing a subscription from the pub/sub domain.

The gateway also creates provisional SIP responses back to the SIP domain based on the status of the pub/sub responses received. Thus the gateway mainly receives messages, reformats them, and forwards them to the proper domain. Messages are associated with client-specific sessions at the gateway. We use the Call Identifier of SIP messages and the subscription identifiers of pub/sub messages to perform this association. These two identifiers are assumed to be unique.

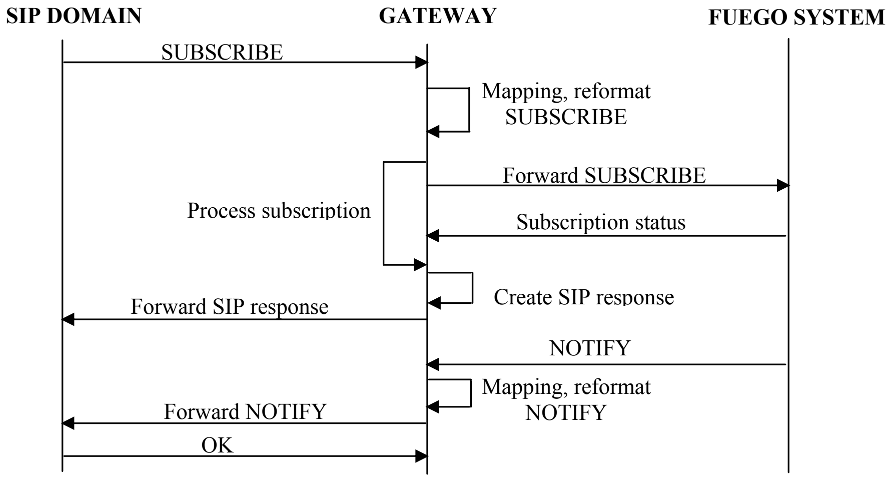


Figure 5: SIP Subscription and Fuego Notification

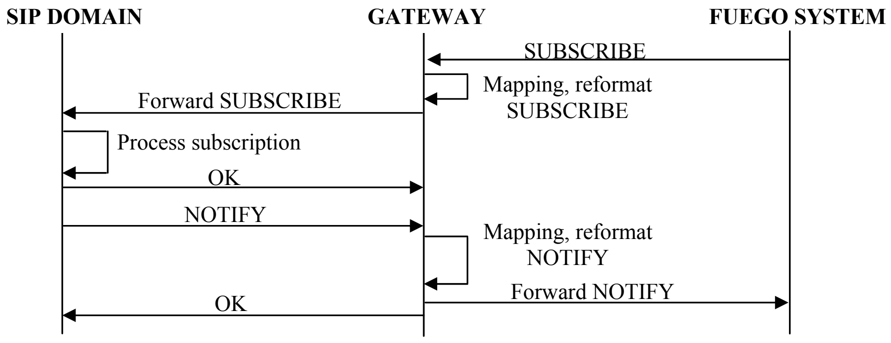


Figure 6: Fuego Subscription and SIP Notification

5.3 Filtering

The default SIP event package does not support filters. The gateway may support them using a proprietary scheme, for example by introducing filters into the message header or payload. Filtering is easy to add for any events to/from the pub/sub domain, because the pub/sub domain natively supports filtering and the filtering data structures may be used by the gateway. If there are any differences in the filtering languages of the two domains appropriate conversion is needed.

Two Internet Engineering Task Force (IETF) drafts about event filtering for SIP events have been proposed. An XML-based filtering format for event notification is described in (Khartabil 2004a). A functional description of event filtering for SIP is given in (Khartabil 2004b). In the latter proposal, the subscriber defines a set of filtering rules in the content part of the SUBSCRIBE message. Filters can be changed within a SIP dialog by sending a new SUBSCRIBE request. The notifier examines the content of the SUBSCRIBE message and sends a 200-class response to the subscriber when the content type is understood and the subscription accepted. Notifications are then sent to the subscriber. A non-200 class response indicates that the subscription was not accepted and created.

5.4 Mobility

The proposed gateway can be extended to support mobility between the two domains. SIP supports different mobility types and several pub/sub systems also support mobility, for example the Fuego system, Siena (Caporuscio 2003), and Rebeca (Mühl 2004). Four different mobility categories need to be examined, namely terminal and user mobility, both intra-and inter-domain. Terminal mobility occurs when the user and the terminal change their location and access point. Terminal mobility for intra-domain operation is supported by both SIP and the mentioned pub/sub technologies by allowing clients to change their access points. Intra-domain mobility can be implemented in a transparent fashion so that applications do not need to be modified. Terminal mobility for inter-domain mobility is more difficult, because the access protocol and API change, which requires that applications support both SIP and pub/sub mechanisms. User mobility happens when the user changes the terminal device. This requires at least support for disconnected operation, but may require support for terminal mobility as well. SIP supports user mobility by forking messages and also re-sending messages, however, longer term disconnections are not supported. Mobility-aware pub/sub systems support disconnected operation by storing undeliverable messages.

Of the discussed mobility types, intra-domain mobility support appears to be more important and interesting for applications than support for mobility between the domains. When a client that uses the gateway moves to the other domain, it no longer needs the gateway service in the direction of the old domain. If the client is interested in events from both domains, the direction of event flow provided by the gateway is reversed. We envisage that event messages from both domains can be buffered at the gateway and that the gateway can provide the necessary APIs for inter-domain mobility management. The roaming client sends a signaling message to the gateway from its current domain. The gateway sends any buffered event messages to the client after receiving the location update. This mechanism needs also additional signaling between clients and the gateway. Gateway-supported mobility in the pub/sub domain needs also to take any existing intra-domain protocols into account.

A trivial solution for inter-domain mobility is to subscribe the same information in both domains using respective mechanisms, receive information only from one domain at a time, and perform duplicate detection in the application. This technique does not require the presence of a gateway at all. The benefit of using a gateway in the inter-domain mobility scenario is that it can perform queue management on behalf of the client. Even if the gateway is used the client needs to support both SIP and the pub/sub API.

5.5 Implementation

We have developed a prototype implementation of the gateway using the JAIN SIP stack (Sun 2003) and the Fuego event system, which are both Java-based systems. We used the NIST open source SIP stack that implements the JAIN SIP API (NIST 2005). The SIP/Fuego event gateway supports the forwarding of event messages across the domains. The current implementation does not support buffering for the pub/sub domain and filtering at the gateway. Filtering is supported at the pub/sub domain, so a SIP client may publish an event that is filtered at the pub/sub domain. Filtering support is relatively easy to add for the SIP domain if the same filtering language is used in both domains. Supporting conversion between different filtering languages is more difficult and, for example, the CORBA Notification Service / JMS interworking bridge does not support filters (OMG 2004).

Buffering is required at the gateway if a requested client is not reachable. In the SIP stack implementation, the proxy retransmits the request to the client and drops it on repeated failures. If the retransmission of a subscription request does not succeed, the SIP notifier removes the subscriber from the subscribers list. The SIP subscriber has to send a new subscription request. The SIP Uniform Resource Identifier (URI) addressing scheme is used to deliver pub/sub messages to the proper SIP clients. Since filtering support is not implemented, the gateway maps events from the pub/sub domain to SIP clients using a specific attribute in the event message. The value of this attribute is the name of the SIP client. The other direction does not require specific mapping, because the pub/sub domain routes messages based on their content or header.

Experimentation with the implemented gateway has concentrated mainly on the scalability and performance of the gateway component. The gateway component was tested by generating and sending a number of subscription and notification messages across the Fuego and SIP event domains. We implemented a simple application that uses the SIP presence package. One version of the application runs in the SIP domain and another version in the Fuego domain. In the experimental scenario the gateway forwards presence updates between the domains.

6. CONCLUSIONS

Interworking between different event systems is required to support communication in heterogeneous environments and gain access to information provided in different technology domains. The challenges of event interworking are differences in semantics, APIs, and the expressiveness of the notification model. In this paper, we have presented the design and implementation of a gateway for interworking SIP event domains with domains using a generic pub/sub API. The SIP event notification differs from the pub/sub model, because it is intended for entity-to-entity communication and it does not support filtering.

The gateway is motivated by the emergence of SIP-enabled devices and the need to access heterogeneous information. The gateway facilitates, for example, the tracking of people and entities from a SIP-enabled device and pushing filtering into the pub/sub domain. In this kind of design, the content-based routing network is the information backbone and SIP is the delivery and control mechanism. The gateway has a simple design, it supports scalability through partitioning, and it may be extended with filtering. Future work includes adding filtering and load balancing support to the gateway system.

REFERENCES

Berger, S. et al, 2003. Ubiquitous Computing Using SIP. *In ACM NOSSDAV 2003*, Monterey, California, USA.

Caporuscio, M. et al, 2003. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *Technical Report CU-CS-944-03*, Department of Computer Science, University of Colorado.

Colouris, G. et al, 1994. *Distributed Systems: Concepts and Design (2nd edition)*. Addison-Wesley, Boston, Massachusetts.

Eugster, P. T. et al, 2003. The many faces of publish/subscribe. *In ACM Computing Surveys (CSUR)*, 35(2), 114 – 131.

IETF, 2005. SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) working group. Available at: http://www.ietf.org/html.charters/simple-charter.html

Khartabil, H. et al, 2004a. An Extensible Markup Language (XML) Based Format for Event Notification Filtering, draft-ietf-simple-filter-format-03. Internet Engineering Task Force.

Khartabil, H. et al, 2004b. Functional Description of Event Notification Filtering, draft-ietf-simple-event-filter-funct-03. Internet Engineering Task Force.

Mühl, G. et al, 2004. Disseminating information to mobile clients using publish/subscribe. *IEEE Internet Computing*, 8(3), 46–53.

NIST, 2005. Open Source SIP stack. National Institute of Standards and Technology. Available at: http://snad.ncsl.nist.gov/proj/iptel/

OMG, 2001. CORBA Notification Service Specification v.1.0. Object Management Group.

OMG, 2004. Notification / Java Messaging Service (JMS) Interworking Specification v.1.0. Object Management Group.

Roach, A., 2002. Session Initiation Protocol (SIP)-specific Event Notification (RFC 3265). Internet Engineering Task Force.

Rosenberg, J. et al, 2002. Session Initiation Protocol (RFC 3261). Internet Engineering Task Force.

Schulzrinne, H., et al., 2000. Application-layer mobility using SIP. Mobile *Computing and Communications Review*, 1(2).

Sun Microsystems, 2003. JSR-000032 JAIN SIP Specification.

Tarkoma, S.,et al., 2005. A data structure for content-based routing. *In Proceedings of EuroIMSA'05*. Grindelwald, Switzerland.

Tarkoma, S. et al, 2003. Client mobility in rendezvous-notify. *In Intl. Workshop on Distributed Event-Based Systems* (DEBS'03).

Call for Papers

4th Workshop on Applications of Wireless Communications (WAWC'06), August 2-4, 2006 – Lappeenranta, Finland

in conjunction with the 15th Summer School on Telecommunications at the Lappeenranta University of Technology, Finland

Important dates

Abstract submission deadline: January 20, 2006
Paper submission deadline: February 20, 2006
Notification of acceptance: March 31, 2006
Revised papers due: May 1, 2006

Overview

WAWC'06 will highlight the latest research and developments in the wireless communication technologies with major emphasis on the applications and services of the wireless communications. This workshop is intended for researchers and professionals interested in the applicability of the wireless communications. This will be a one-day workshop featuring invited presentations and refereed paper presentations. It will be held in conjunction with the 15th Summer School on Telecommunications that concentrates on Cross-Technical issues of the Wireless World. The program also includes a 24h programming event, Code Camp.

Scope/Topics

Applications and Services on wireless communication should concentrate at one of the following topics.

- Personal Mobility
- Personalization
- Security, Trust and Privacy
- Semantic Interoperability
- Context related issues
- Adaptation
- Generic Service Elements
- Orchestration
- Composability
- Management
- Ambient Awareness
- Pervasive Communications
- Ubiquitous Communications
- Wireless applications and services
- Conceptualization issues of Applications
- Open Platforms
- User-Centric Approaches

WAWC'06 is NOT an appropriate forum for research focused narrowly on the physical, link or network layers. All papers are supposed to be related to the applications and services of the given topics.

Please feel free to contact the program chair at WAWC@lut.fi to determine the appropriateness of the topic. By sending abstract the program committee can control the appropriateness of the topics to be submitted as full papers or posters.

What to submit

Submission takes place in two phases; at first an abstract is submitted and the program committee will inform the sender the appropriateness of the topic of sent abstract. After the appropriateness is validated, the submission can be either full paper (maximum of 10 single-spaced A4 pages, including figures, tables and references using point 12 font) or poster (maximum of 5 single-spaced A4 pages, including figures, tables and references using point 12 font).

A good paper demonstrates that

- Application area presented is or will be of greater interest of majority
- Wireless communication has a significant role in the solution
- The realization of the application presents a new way of doing things

Papers may be selected as:

- Full papers (max. 10 pages) or Posters (max. 5 pages)

WAWC'06 among most conferences and journals requires that papers must be original and published only in this conference. Proceedings will be published in the Acta Universitatis Lappeenrantaensis series of Lappeenranta University of Technology.

All selected papers must be paid in advance and presented in the conference.

How to submit

All submissions for WAWC'06 will be electronic, in PDF format. Please, submit papers by sending them to WAWC@lut.fi. Authors will be notified of the receipt of submission via email. If you do not receive notification, contact WAWC@lut.fi.

Registration

Complete program and registration information will be available in May 2006 on the Workshop Web site http://www.it.lut.fi/WAWC. The information will be in both html and a printable PDF file formats.

Questions

Inquiries concerning the workshop to WAWC@lut.fi.

Organized by the Laboratory of Communications Engineering Lappeenranta University of Technology.

ISBN 952-214-057-0
ISBN 952-214-058-9 (PDF)
ISSN 1456-4491
Lappeenranta 2005