Self-Adaptive Communication Mechanisms for Cooperative Information Networks

Submitted to
Graduate School of Information Science and Technology
Osaka University

July 2008

Yoshiaki TANIGUCHI

List of Publications

Journal Papers

1. Y. Taniguchi, N. Wakamiya, and M. Murata, "A traveling wave based communication mechanism for wireless sensor networks," *Journal of Networks (JNW)*, Vol. 2, No. 5, pp. 24–32, Sept. 2007.

2. Y. Taniguchi, N. Wakamiya, and M. Murata, "A proxy caching system for MPEG-4 video streaming with a quality adaptation mechanism," *WSEAS Transactions on Communications*, Vol. 6, No. 10, pp. 824–832, Oct. 2007.

3. Y. Taniguchi, N. Wakamiya, and M. Murata, "Quality-aware cooperative proxy caching for video streaming services," submitted to *Journal of Networks (JNW)*, Apr. 2008.

Refereed Conference Papers

1. Y. Taniguchi, A. Ueoka, N. Wakamiya, M. Murata, and F. Noda, "Implementation and evaluation of proxy caching system for MPEG-4 video streaming with quality adjustment mechanism," in *Proceedings of the 5th AEARU Workshop on Web Technology*, pp. 27–34, Oct. 2003.

2. Y. Taniguchi, N. Wakamiya, and M. Murata, "Implementation and evaluation of cooperative proxy caching mechanisms for video streaming services," in *Proceedings of SPIE's International Symposium on the Convergence of Information Technologies and Communications (ITCom 2004)*, pp. 288–299, Oct. 2004.

3. Y. Taniguchi, N. Wakamiya, and M. Murata, "A distributed and self-organizing data gathering scheme in wireless sensor networks", in *Proceedings of the 6th Asia-Pacific*

Symposium on Information and Telecommunication Technologies (APSITT 2005), pp. 299–304, Nov. 2005.

4. Y. Taniguchi, N. Wakamiya, and M. Murata, "A self-organizing communication mechanism using traveling wave phenomena for wireless sensor networks," in *Proceedings of the 2nd International Workshop on Ad Hoc, Sensor and P2P Networks (AHSP 2007)*, pp. 562–569, Mar. 2007.

5. Y. Taniguchi, N. Wakamiya, and M. Murata, "A communication mechanism using traveling wave phenomena for wireless sensor networks," in *Proceedings of the 1st Annual IEEE International Workshop: From Theory to Practice in Wireless Sensor Networks (t2pWSN 2007)*, June 2007.

6. Y. Taniguchi, N. Wakamiya, and M. Murata, "Demo abstract: a traveling wave-based self-organizing communication mechanism for WSNs," in *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys 2007)*, pp.399-400, Nov. 2007.

7. Y. Taniguchi, N. Wakamiya, M. Murata, and T. Fukushima, "An autonomous data gathering scheme adaptive to sensing requirements for industrial environment monitoring," submitted to *the 2nd International Conference on New Technologies, Mobility and Security (NTMS 2008)*, June 2008.

Non-Refereed Technical Papers

1. Y. Taniguchi, M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara, "Implementation and evaluation of proxy caching mechanisms with video quality adjustment," *Technical Report of IEICE (CQ2002-72)*, Vol. 102, No. 191, pp. 41–46, July 2002. (in Japanese)

2. Y. Taniguchi, A. Ueoka, N. Wakamiya, M. Murata, and F. Noda, "Implementation and evaluation of proxy caching system for MPEG-4 video streaming with quality adjustment mechanism," *Technical Report of IEICE (NS2003-45)*, Vol. 103, No. 122, pp. 45–48, June 2003. (in Japanese)

3. Y. Taniguchi, N. Wakamiya, and M. Murata, "Implementation and evaluation of cooperative proxy caching system for video streaming services," *Technical Report of*

IEICE (IN2003-190), Vol. 103, No. 650, pp. 13–18, Feb. 2004. (in Japanese)

4. Y. Taniguchi, N. Wakamiya, and M. Murata, "A distributed and self-organizing communication mechanism based on traveling wave phenomena for wireless sensor networks," *Technical Report of IEICE (NS2006-48)*, Vol. 106, No. 167, pp. 17–20, July 2006. (in Japanese)

5. Y. Taniguchi, N. Wakamiya, and M. Murata, "Implementation and evaluation of traveling wave based communication mechanism for wireless sensor networks," *Technical Report of IEICE (NS2007-40)*, Vol. 107, No. 146, pp. 1–6, July 2007. (in Japanese)

6. Y. Taniguchi, N. Wakamiya, M. Murata, and T. Fukushima, "A traveling wave based data gathering scheme adaptive to sensing requirements," to be presented at *IEICE USN Workshop*, July 2008. (in Japanese)

Preface

The Internet has evolved into an important social infrastructure over the recent years. Future information networks are expected to offer a wide variety of network services, for instance, by allowing various types of devices to anytime and anywhere interact and connect to a network topology. In such an environment, the number of nodes and their locations, the amount of traffic and their communication patterns, the availability and quality of communication links, the application/user requirements on network services, as well as their behavior become unpredictable and subject to dynamic changes. Therefore, future information networks should be adaptive to the heterogeneity of connected devices and dynamic changes in the network environment. Furthermore, the number of connected devices may be very large, which makes a centralized control and management of the entire network impossible. Thus, it is essential that cooperation among nodes is performed by taking only local exchange of information into account. Therefore, the communication mechanism in future cooperative networks should operate in an adaptive and fully-distributed, i.e. self-adaptive, manner.

In this thesis, we focus on two very different types of information networks to illustrate the need for adaptation from various perspectives. In our first scenario, we consider a video streaming system that is adaptive to the dynamically changing requirements on the quality of video data provided to heterogeneous users in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality. Video servers in video streaming systems should operate cooperatively to provide users with a continuous and high-quality video streaming service. The second scenario considers wireless sensor networks that are very limited in their processing capabilities, but must be adaptive to various and dynamically changing communication patterns and frequencies of data gathering. Sensor nodes in wireless sensor networks should cooperate to gather or diffuse information in an energy-efficient, robust, fully-distributed, and self-organizing manner.

We begin this work by proposing, designing, and implementing a video proxy caching system that serves as the basis for our further investigations. An adaptation scheme of the video quality through the proxy caching system is implemented to provide heterogeneous clients with a continuous and high-quality video streaming service. The proxy caching system is then evaluated for an MPEG-4 video streaming service, taking into account the limitations in commercial and widely used client/server applications by employing off-the-shelf and common hardware and software for the server and clients. The practicality of our proposed system is verified through experimental evaluations. Furthermore, it is shown that our proxy caching system can dynamically adapt the quality of video streams to the network conditions while providing users with a continuous and high-quality video streaming service. We also verify that our system can provide 50 clients with smooth video streaming while the CPU usage of the proxy is less than 8%.

In order to provide a more scalable, efficient, and high-quality video streaming service, we extend the caching mechanism by taking cooperation among proxies into account. The mechanism itself consists of three parts: block provisioning, block prefetching, and cache replacement. The main benefits of our proxy cooperation mechanism include a reduction of the perceived network latency and a higher degree of content availability through cooperative caching among proxies. Simulation experiments verify that our proposed cooperative mechanism can provide users with lower delays and higher quality video streaming services in comparison with the independent caching mechanism. In addition, we implement the proposed collaborative proxy caching mechanism in our previously established prototype environment of a real system for MPEG-4 video streaming. Through experimental evaluations, it is shown that by introducing collaborative behavior, our proxy caching system is improved in terms of providing users with a continuous video streaming service even under dynamically changing network conditions.

The second part of this thesis deals with wireless sensor networks, which are an entirely different type of information networks compared to the high-performance video streaming system. To emphasize that especially devices with low processing abilities can benefit from collaborative actions, we consider a mechanism adaptive to various types of communication. In our case, two different communication strategies, diffusion of information from a source node to all other nodes and gathering of data from all nodes to a specific sink node are considered, as they represent the most common cases found in sensor network

applications. Both forms of communication require a coordinated information dissemination strategy, basically forming a kind of traveling wave. This is realized by a theoretical model of pulse-coupled oscillators that stimulate or inhibit each other by local interactions through firing of pulses. We study the conditions that lead to a desired form of traveling wave regardless of the initial phase of the oscillators and propose a fully-distributed, self-organized communication mechanism for wireless sensor networks. Through simulation experiments, we confirm that our scheme delivers sensor information to/from designated nodes in a more robust and energy-efficient manner than other methods. Our mechanism can extend the network lifetime by a factor of up to 12.8 compared to the directed diffusion method, which is a well-known reference model found in the lifetime. However, due to the underlying biological model, we experience a slight delay until the data dissemination process is completed. Implementations using commercial MICAz sensor units give further evidence of the applicability of our approach. Through experimental evaluations, we confirm that data delivery ratio of about 95 % is accomplished where 16 nodes are located in a 4×4 grid layout.

Finally, we study the timing intervals of sensing and data gathering periods in more detail, since this interval is greatly influenced by the application requirements and environmental conditions of the wireless sensor network. We propose a data gathering mechanism that is adaptive to the different sensing requirements in networks composed of sensor nodes with multiple sensing capabilities. To accomplish a self-organizing and collaborative control, we adopt the response threshold model for adaptive sensing task engagement and the pulse-coupled oscillator model for energy-efficient transmission and sleep scheduling in an industrial application scenario. Through simulation experiments, we confirm that autonomous and energy-efficient data gathering can be accomplished well by a collaborative scheme in spite of dynamically changing sensing requirements. In the simulation scenario, our mechanism can further reduce the energy consumption per node.

Acknowledgments

First of all, I would like to express my sincere appreciation to my supervisor, Professor Masayuki Murata, Graduate School of Information Science and Technology, Osaka University, for his encouragement, valuable discussions, and meaningful advices. He introduced me to the area of information network and has inspired me to aim for higher goals.

I would like to thank to Professor Koso Murakami, Professor Makoto Imase, and Professor Teruo Higashino, Graduate School of Information Science and Technology, Osaka University, and Professor Hirotaka Nakano, Cybermedia Center, Osaka University, for their valuable comments, technical guidances, and reviewing of this thesis.

I greatly acknowledge Associate Professor Naoki Wakamiya, Graduate School of Information Science and Technology, Osaka University, for his critical comments, unerring guidance, and continuous support. This work would not have been possible without him.

I would also thank to President Hideo Miyahara, National Institute of Information and Communications Technology, Associate Professor Masashi Sugano, Osaka Prefecture University, Associate Professor Ken'ichi Baba, Associate Professor Hiroyuki Ohsaki, Associate Professor Go Hasegawa, Osaka University, Associate Professor Shingo Ata, Osaka City University, Assistant Professor Shin'ichi Arakawa, Assistant Professor Masahiro Sasabe, Assistant Professor Yuichi Ohshita, Osaka University, for their helpful advices and comments of this work.

I also wish to thank Mr. Atsushi Ueoka and Mr. Fumio Noda, Systems Development Laboratory, Hitachi, Ltd., for their help, encouragement and suggestions for the implementation of the proxy caching system for MPEG-4 video streaming services in this thesis. I would also like to thank Dr. Takashi Fukushima, Kobe Steel, Ltd., for his valuable comments and helps from industrial viewpoint in wireless sensor networks.

Thank you to all of past and present professors, researchers, postdoctorals, secretaries,

and students of Advanced Network Architecture Laboratory in Graduate School of Information Science and Technology, Osaka University, for their detailed and valuable advices, suggestions, and encouragement. Specially, I would like to thank Specially Appointed Associate Professor Kenji Leibnitz for his suggestions, encouragement, corrections of this work.

I wish to thank Assistant Professor Tomoya Kitani, Assistant Professor Yoshitaka Nakamura, Nara Institute of Science and Technology, Associate Professor Hiroyuki Hisamatsu, Osaka Electro-Communication University, and all my friends for their suggestions and encouragement during my undergraduate and graduate life.

Finally, I deeply thank my parents and my elder sister for their invaluable support and encouragement in my life. This work would not have been possible without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The Internet has evolved into an important social infrastructure over the recent years. Future information networks are expected to offer a wide variety of network services, for example, environmental monitoring, video streaming services, seamless internet access, building automation, by allowing various types of devices to anytime and anywhere interact and connect to a network topology. In such an ambient environment [1], the number of nodes and their locations, the amount of traffic and their communication patterns, the availability and quality of communication links, the application/user requirements on network services, as well as their behavior become unpredictable and subject to dynamic changes. Therefore, future information networks should be adaptive to the heterogeneity of connected devices and dynamic changes in network environment [2, 3].

Traditional adaptive mechanisms are pre-programmed and pre-installed into a system taking into account possible conditions predicted at the deployment phase, and achieve good performance within their predicted range. However, once unexpected events happen or the operation conditions go out of the intended range, the system may easily collapse. Since it is expected that future information networks are likely to face unexpected events more frequently than before due to the introduction of new technologies, a robust communication mechanism has to achieve higher adaptability by being able to cope with the changing environment [3].

Furthermore, since traditional client/server architectures lack scalability, distributed and cooperative networks such as P2P networks, wireless ad hoc networks, and sensor

networks have gathered much attention and are expected to be widely used in future information networks. In such networks, the number of connected devices/users may be very large, which makes a centralized control and management of the entire network impossible. Thus, it is essential that cooperation among nodes is performed by taking only local exchange of information into account. Therefore, the communication mechanism in future cooperative networks should operate in an adaptive and fully-distributed, i.e. *self-adaptive*, manner.

To contribute to the development of future cooperative information networks, we mainly focus in this thesis on the principle of self-adaptive control. A lot of research has been dedicated to self-adaptive control mechanisms for cooperative information networks. For example, in routing mechanisms in mobile ad hoc networks [4, 5], nodes may join or leave the network or move to other locations while sending or receiving data. To maintain communication among each other, the routing mechanism must be adaptive to the location and behavior of the nodes. On the other hand, in wireless sensor networks, coverage control mechanisms [6] must adapt their sensing behavior to efficiently monitor their surrounding target region through cooperation among nodes and without any centralized control.

Biologically-inspired approaches are known to have self-adaptive capabilities [2, 7]. It is known that biological systems have the inherent feature of self-adaptability, although they are rather slow to adapt to environmental changes [8, 9]. Some papers adopt biologically-inspired approaches to achieve self-adaptive control in cooperative information networks [10-12]. For example, [11] considers multi-path routing in overlay networks. The probability of path selection is adaptive to the current environment in terms of available bandwidth and latency by using attractor selection that is based on a biological model.

Since there are many types of information networks and different dynamic factors as described above, we focus on two very distinct types of information networks, i.e. *video streaming systems* and *wireless sensor networks*, in this thesis to illustrate the need for self-adaptation from various perspectives.

Video Streaming Systems

In video streaming systems, a client receives a video stream from a video server over the Internet and plays each block of the stream as it gradually arrives. With the increase in computing power and the proliferation of the Internet, video streaming systems have become widely deployed [13-15].

In the video streaming systems, requirements on the quality of video data provided to clients are heterogeneous and dynamically changing, for the heterogeneity in clients in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality. Since there is no guarantee on bandwidth, delay, or packet loss probability in the current Internet, which only provides best-effort packet delivery, it is difficult to provide clients with continuous or reliable video streaming. Furthermore, most of today's streaming system is based on client/server architecture, which lack scalability against increased clients.

To tackle the scalability problem, introducing cooperative intermediate nodes, i.e. *proxies*, near the clients in the system, is effective and enables low-delay and high-quality video streaming. In addition, by introducing adaptive video quality capabilities, i.e. video filters, at the proxies, the system can handle heterogeneous and dynamically changing requirements and provide users with a continuous and high-quality video streaming service.

There have been several publications on proxy caching mechanisms with capability of video quality adaptation for video streaming systems [16-23]. However, they do not consider cooperation among proxies, lack the scalability to rate and quality variations, or assume proprietary and specially designed server/client applications which are not widely available.

Wireless Sensor Networks

A wireless sensor network is constituting of battery-powered sensor nodes to obtain information on behavior, condition, and position of elements in a local or remote region [7, 24-30]. Each node in such a sensor network has a processor with limited computational capability, small memory, and radio transceiver.

Due to several restrictions including limited battery capacity, random deployment, and large number of fragile sensor nodes, wireless sensor networks should have cooperative behavior and a mechanism should be energy-efficient, adaptive, robust, fully-distributed, and self-organizing. Furthermore, application requirements on communication patterns, sensing frequencies, and the number of nodes to monitor and report the phenomena are various and dynamically changing.

As an example of dynamic change in communication pattern, a sensor node detecting an emergency would distribute the information over the whole sensor network to alert the other nodes and make them cooperatively react to the emergency whereas all sensor nodes report sensing data to a central node, called sink, at regular intervals in case without

emergency. On the contrary, a sensor node detecting an uncertain condition would collect and aggregate sensor information of other neighboring nodes to have a more precise view of the environment by conjecturing from the collected information.

As another example of cases where the number of nodes to monitor and report the phenomena should be regulated in accordance with its criticality and importance in this thesis, we consider temperature and CO gas sensors deployed on the surface of shaft furnace of a steel plant for industrial environment monitoring. Temperature changes slowly in the order of hours and once it increases, it stays high for a long period of time. Therefore, nodes are required to monitor temperature more frequently when temperature changes are detected, while they can decrease the sensing frequency under stable conditions. On the contrary, CO gas would suddenly leak and spread fast. Therefore, nodes are required to monitor CO gas more frequently than temperature while CO gas exists.

To adapt to various types of communication patterns in accordance with application requirements, sensing frequencies, and the number of nodes to monitor and report the phenomena, sensor nodes should adapt their behavior in a fully-distributed and self-adaptive manner.

1.2 Outline of Thesis

In this thesis, we propose self-adaptive communication mechanisms for video streaming systems and wireless sensor networks where servers or nodes operate in a cooperative way to satisfy heterogeneous and dynamically changing application/user requirements in a heterogeneous and dynamically changing environment. Each of these mechanisms is described together with simulation or experimental results for evaluations.

A Proxy Caching System with Quality Adaptation for MPEG-4 Video Streaming Services [31-33]

We first consider a video streaming systems scalable to number of clients and adaptive to requirements on the quality of video data. To accomplish low-delay and high-quality video distribution without imposing extra load on the system, we adopt as proxy mechanism widely used in WWW systems.

In Chapter 2, we begin this work by proposing, designing, and implementing a video proxy caching system that serves as the basis for our further investigations. In our proposed

system, a video stream is divided into blocks so that the cache buffer and the bandwidth can be used efficiently. A proxy retrieves a block from the server, deposits it in its local cache buffer, and provides requesting clients with blocks in time. It maintains the cache with a limited capacity by replacing unnecessary cached blocks with a newly retrieved block. A proxy cache server prefetches video blocks that are expected to be required in the near future to avoid cache misses. It also adjusts the quality of a cached or retrieved video block to the appropriate level through video filters to handle client-to-client heterogeneity without involving the original video server.

We build a prototype of our proxy caching system for MPEG-4 video streaming services [34] on a real current system. We employ off-the-shelf and common applications for server and client programs to implement our system. On the contrary to other proposals, our system does not need any modifications in server/client applications ore systems as far as they conform to streaming standards. Our proxy caching system can be applied to environments in which RTSP/TCP is used to control video streaming and RTP/UDP to deliver them. We introduce a TFRC (TCP Friendly Rate Control) [35] mechanism to the system for video streaming to share the bandwidth fairly with conventional TCP sessions. We use a frame dropping filter to adapt the rate of video streams to the bandwidth available to service. Through experimental evaluations, it is shown that our proxy caching system can provide users with a continuous and high-quality video streaming service by introducing video quality adaptation capability.

A Cooperative Proxy Caching Mechanism with Quality Adaptation for Video Streaming Services [36-41]

Although the video streaming system proposed in Chapter 2 can provide users with a continuous and high-quality service, the system is intended for only a single proxy. A proxy always retrieves blocks from a video server at cache miss although there are other proxies having low-delay and broadband links to the proxy.

In Chapter 3, in order to provide a more scalable, effective, and high-quality video streaming service, we extend our caching mechanism by considering cooperation among proxies. The new self-adaptive mechanism consists of three parts: block provisioning, block prefetching, and cache replacement. The main benefits of our proxy cooperation mechanism include reducing the perceived network latency and achieving a higher degree of content availability by cooperative caching among proxies. Through simulation experiments, it is

shown that our proposed cooperation mechanism can provide users with low-delay and high-quality video streaming services in comparison with independent caching mechanism. In addition, to verify the practicality and adaptability of our proposal to existing video streaming services, we implement our cooperative proxy caching mechanism on a real system for MPEG-4 video streaming services extending our implemented system described in Chapter 2. Through experimental evaluations, it is shown that our cooperative proxy caching system can provide users with a continuous video distribution under dynamically changing network conditions.

A Traveling Wave-based Communication Mechanism Adaptive to Application Requirements for Wireless Sensor Networks [42-48]

In Chapter 4, we change our attention to wireless sensor networks and we consider a self-organizing communication mechanism adaptive to application requirements.

In wireless sensor networks, a communication mechanism should be able to handle various types of communication, i.e. diffusion and gathering, involving the whole network in accordance with dynamically application requirements. To answer dynamically changing application requirements, a communication mechanism should handle both types of communication, especially in an self-adaptive and self-organizing manner. For this purpose, we adopt a pulse-coupled oscillator (PCO) model based on biological mutual synchronization such as that observed in flashing fireflies [49-53]. In a PCO model, synchronous behavior of a group of oscillators is considered. Each oscillator operates on a timer. When the phase of the timer reaches one, an oscillator fires. Oscillators coupled with the firing oscillator are stimulated and they shift the phase of timers by a small amount. Through mutual interactions by stimuli among oscillators, the global synchronization where all oscillators fire synchronously, or a traveling wave, where oscillators behave synchronously keeping fixed phase difference, appears.

In this chapter, in contrast to the other work [54-65], we focus on traveling wave phenomena in a PCO model. By adjusting parameters and functions of a PCO model, we can control the frequency, form, and direction of a wave. We first investigate conditions of a phase response curve (PRC) with which a wireless sensor network reached a preferred phase-lock condition where the phase differences among sensor nodes are kept constant from arbitrary settings of the initial phase of sensor nodes. Then, we propose a self-adaptive and self-organizing communication mechanism which generated concentric traveling waves

centered at a sensor node, which wanted to gather information from all sensor nodes or diffuse information to all sensor nodes. Through simulation experiments, we confirm that our scheme delivers sensor information to/from designated nodes in a more energy-efficient manner than other method, although it takes time to generate a traveling wave. Furthermore, we implement our mechanism using commercial wireless sensor units, MICAz [66]. Since collisions among synchronized packet emissions affects the performance, we extend the mechanism to distribute timing of packet emission and we confirm that data delivery ratio of about 95 % is accomplished.

A Data Gathering Mechanism Adaptive to Sensing Requirements for Wireless Sensor Networks [67, 68]

In some class of applications, a node need to change its sensing frequency to monitor the region or target more frequently when it detects unusual condition and phenomena. Furthermore, the number of nodes which monitor and report the phenomena should be regulated in accordance with its criticality and importance.

In Chapter 5, to tackle the above-mentioned problem, we propose a data gathering mechanism adaptive to dynamically changing sensing requirements. In our mechanism, nodes operate on traveling wave-based periodic data gathering at regular data gathering intervals as described in Chapter 4. We assume that the regular sensing frequency is the same among sensors of different types. When condition of sensing target, such as temperature and gas concentration, changes at a certain point, surrounding nodes decide whether to monitor the target more frequently or not depending on the need for sensing. To regulate the number of nodes engaged in frequent monitoring of target in a self-organizing way, we adopt a response threshold model [69], i.e. a mathematical model of division of labor. Once a node considers to monitor the target more frequently, it changes its sensing frequency higher. So that obtained sensor data are forwarded to a base station at the higher frequency, nodes on the path to the base station also adapt to the new frequency. As a result, two or more traveling waves emerge in part or as a whole in a wireless sensor network. Through simulation experiments, we confirm that autonomous and energy-efficient data gathering can be accomplished satisfying dynamically changing sensing requirements in a energy-efficient manner.

Chapter 2

A Proxy Caching System with Quality Adaptation for MPEG-4 Video Streaming Services

2.1 Introduction

With the increase in computing power and the proliferation of the Internet, video streaming services have become widely deployed. Through these services, a client receives a video stream from a video server over the Internet and plays it as it gradually arrives. However, on the current Internet, only the best effort service, where there is no guarantee on bandwidth, delay, or packet loss probability, is still the major transport mechanism. Consequently, video streaming services cannot provide clients with continuous or reliable video streams. As a result, the perceived quality of video streams played at the client cannot satisfy expectations, and freezes, flickers, and long pauses are experienced. Furthermore, most of today's Internet streaming services lack scalability against increased clients since they have been constructed on a client/server architecture, e.g., YouTube [13], Google Video [14], GyaO [15], and so on. A video server must be able to handle a large number of clients, which have diverse requirements on a received video stream for their heterogeneity.

Proxy mechanisms widely used in WWW systems offer low-delay and scalable delivery of data by means of a "proxy server". A proxy server caches multimedia data that has passed through its local buffer, called the "cache buffer", and it then provides the cached data to

users on demand. By applying this proxy mechanism to video streaming systems, low-delay and high-quality video distribution can be accomplished without imposing extra load on the system [16-20]. In addition, the quality of cached video data can be adapted appropriately at a proxy to support heterogeneous clients in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality [21-23]. There have been proposals for proxy caching mechanisms for video streaming services. However, they do not consider the client-to-client heterogeneity, lack the scalability or adaptability to rate and quality variations, or assume specially designed server/client applications which are not widely available. We also proposed a proxy mechanism with the quality scaling capability for MPEG-2 video streaming services and evaluated its effectiveness through simulation and prototype-based experiments [70]. However, as the others, we had to build all of a server, a proxy, and clients to realize their mechanisms.

In this chapter, we propose and design a proxy caching system for MPEG-4 video streaming services to attain high-quality, continuous, and scalable video distribution. In our system, a video stream is divided into blocks so that the cache buffer and the bandwidth can be used efficiently. A proxy retrieves a block from the server, deposits it in its local cache buffer, and provides requesting clients with blocks in time. It maintains the cache with a limited capacity by replacing unnecessary cached blocks with a newly retrieved block. A proxy cache server prefetches video blocks that are expected to be required in the near future to avoid cache misses. It also adjusts the quality of a cached or retrieved video block to the appropriate level through video filters to handle client-to-client heterogeneity without involving the original video server. These mechanisms are based on our previous proposal [70], but we modify and improve them taking into account limitations in commercial and widely used server/client applications.

We build a prototype of our proxy caching system for MPEG-4 video streaming services on a real current system. We employ off-the-shelf and common applications for server and client programs to implement our system. On the contrary to other proposals, our system does not need any modifications in server/client applications or systems as far as they conform to streaming standards. Our proxy caching system can be applied to environments in which RTSP/TCP is used to control video streaming and RTP/UDP to deliver them. We introduce a TFRC (TCP Friendly Rate Control) [35] mechanism to the system for video streaming to share the bandwidth fairly with conventional TCP sessions. We use a frame dropping filter to adapt the rate of video streams to the bandwidth available to service.

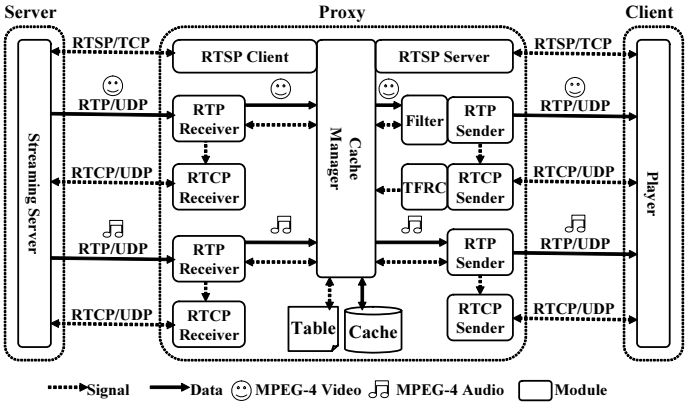


Figure 2.1: Modules constituting system

Through evaluations from several performance points of view, we prove that our proxy caching system could dynamically adjust the quality of video streams to fit to network conditions while providing users with a continuous and high-quality service. We also verify that our system can provide 50 clients with smooth video streaming.

The rest of this chapter is organized as follows. In Section 2.2, we describe our proxy caching system and explains how it is implemented, and conduct several experiments to verify the practicality of our system in Section 2.3. Finally, we conclude this chapter in Section 2.4.

2.2 A Proxy Caching System with Video Quality Adaptation

Figure 2.1 illustrates modules that constitute our proxy cache server. Video streaming is controlled through RTSP/TCP sessions. There are two sets of sessions for each client. The first is established between an originating video server and a proxy to retrieve uncached blocks. The other is between the proxy and the client. Each of video and audio stream is transferred over a dedicated RTP/UDP session. The condition of streaming is monitored over RTCP/UDP sessions. A proxy server has a cache to deposit video data and a filter to adapt the quality of video to the TCP-friendly rate [35]. A video stream is coded using an MPEG-4 video coding algorithm, and it is compliant with ISMA 1.0 [34]. In this thsis, we employed the *Darwin Streaming Server* [71] as a server application, and *RealOne Player* [72] and *QuickTime Player* [73] as client applications. However, other server or

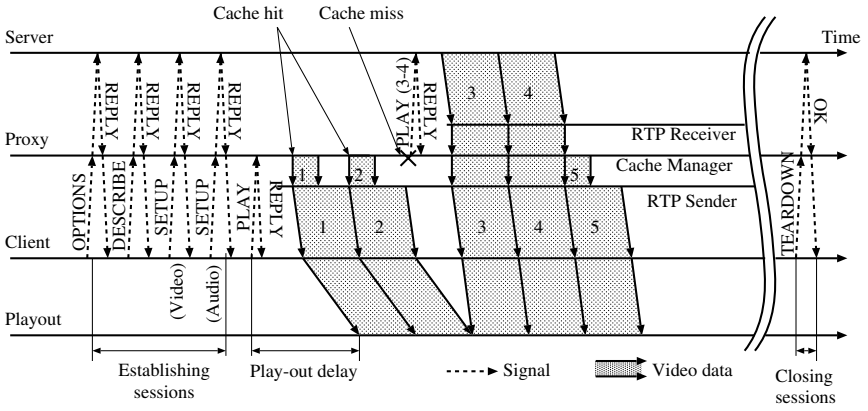


Figure 2.2: Basic behavior of our proxy caching system

client applications being compliant with standard [34] with no or small modification. If other coding algorithm, e.g. MPEG-2, is used, the filtering module is only needed to be changed, which adapts the video rate by manipulating the video data, as far as server and client applications employ a set of the standard protocols, i.e. RTP/UDP, RTCP/UDP, and RTSP/TCP.

2.2.1 Basic Behavior

Figure 2.2 illustrates the basic behavior of our system. In the proxy cache server, a video stream is divided into blocks so that the cache buffer and the bandwidth can be efficiently used. Each block corresponds to a sequence of VOPs (Video Object Planes) of MPEG-4. A block consists of a video block and an audio block, and they are separately stored. The number of VOPs in a block is determined by taking into account the overhead introduced in maintaining the cache and transferring data block-by-block. The strategy used to determine the block size is beyond the scope of this thesis. We used 300 in our empiric implementation. Since an MPEG-4 video stream is coded at 30 frames per second, a block corresponds to ten seconds of video. Segmentation based on VOP was reasonable since packetization based on this is recommended in RFC3016 [74]. Furthermore, we could use the range field of the RTSP PLAY message to specify a block, e.g. `Range 20-30`, because we could easily specify the time that the block corresponded to.

First, a client begins by establishing connections for audio/video streams with the proxy server using a series of RTSP OPTIONS, DESCRIBE, and SETUP messages. An OPTIONS

message is used to request communication options. A DESCRIBE message is used for media initialization and a SETUP message is used for transport parameter initialization. These RTSP messages are received by the *Cache Manager* through an *RTSP Server* module. The proxy server relays these RTSP messages to the video server. Thus, connections between the video server and the proxy server are also established at this stage. Then, the client requests delivery of the video stream by sending an RTSP PLAY message. When a connection between the video server and the proxy server is not used for the predetermined timeout duration, the video server terminates the connection according to RTSP specification. In our system, the proxy server maintains the connection for future use by regularly sending an RTSP OPTIONS message after 90 seconds idle period.

A proxy maintains information about cached blocks in the *Cache Table*. Each entry in the table contains a block identifier, the size of the cached block, and the flag. The size is set at zero when the block is not cached. The flag is used to indicate that the block is being transmitted. On receiving a request for a video stream from a client through the *RTSP Server*, the *Cache Manager* begins providing the client with blocks. The request is divided into blocks, and *Cache Manager* examines the table every interval of the block. If the requested block is cached, i.e. cache hit, the *Cache Manager* reads it out and sends it to the *RTP Sender*. The *RTP Sender* packetizes the block and sends the packet to the client on time. The quality of video blocks is adapted to fit the bandwidth on the path to the client by the *Filter*. Our proxy cache server adjusts the video rate to the bandwidth estimated by the TFRC module to share the bandwidth among video sessions and conventional data sessions in a friendly and fair manner.

When a block is not cached in the local cache buffer, the *Cache Manager* retrieves the missing block by sending an RTSP PLAY message to the video server. To use bandwidth efficiently, and prepare for potential cache misses, it also requests the video server to send succeeding blocks that are not cached, by using the range field of the RTSP PLAY message. Blocks 3 and 4 in Fig. 2.2 have been retrieved from the video server by sending one RTSP PLAY message. Block identifiers are indicated beside the PLAY message in Fig. 2.2 for easier understanding.

On receiving a block from the video server through the *RTP Receiver*, the *Cache Manager* sets its flag to ON to indicate that the block is being transmitted, and it relays the block to the *RTP Sender* VOP by VOP. When reception is completed, the flag is cancelled

and the *Cache Manager* deposits the block in its local cache buffer. However if the retrieved block is damaged by packet loss, the *Cache Manager* doesn't deposit it. If there is not enough room to store the newly retrieved block, the *Cache Manager* replaces one or more less important blocks in the cache buffer with the new block.

A client receives blocks from a proxy and first deposits them in a so-called play-out buffer. Then, after some period of time, it gradually reads blocks out from the buffer and plays them. By deferring the play-out as illustrated in Fig. 2.2, a client can prepare for unexpected delay in delivery of blocks due to network congestion or cache misses.

When a proxy server receives an RTSP TEARDOWN message from a client, the proxy server relays the message to the video server, and closes the sessions.

2.2.2 Block Retrieval

When a requested block is not cached in the local cache buffer, the *Cache Manager* should retrieve the block. Since we adopt an off-the-shelf application for the video streaming server, it cannot adjust the quality of video blocks. Therefore, in our system, the *Cache Manager* always retrieves the missing block with the highest quality, i.e. the quality with which the video stream was coded, from the video server. Of course, when we have a video server capable of quality adaptation, our proposed scheme can attain more efficient and effective control [70].

2.2.3 Rate Control with TFRC

There is a variety of mechanisms to measure the available bandwidth [75-78]. However, it introduces additional traffic or causes delay to a video session. In addition, a selfish behavior of a video session to occupy the whole available bandwidth would affect the others and consequently the performance of network deteriorates. Therefore, our proxy cache server adjusts the video rate to the bandwidth estimated by the TFRC.

TFRC enables a non-TCP session to behave in a TCP-friendly manner. The TFRC sender estimates the throughput of a TCP session sharing the same path using following equation.

$$X = \frac{s}{R\sqrt{\frac{2bp}{3}} + t_{RTO}(3\sqrt{\frac{3bp}{8}})p(1+32p^2)}, \tag{2.1}$$

where X is the transmit rate in bytes/second. s is the packet size in bytes. R is the round

trip time in seconds. p is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted. t_{RTO} is the TCP retransmission timeout value in seconds. b is the number of packets acknowledged by a single TCP acknowledgment.

In the system we implemented, those information are obtained by exchanging RTCP messages between the *RTCP Sender* of the proxy cache server and the client application. A client reports the cumulative number of packets lost and the highest sequence number received to a proxy. From those information, the proxy obtains the packet loss probability. RTT is calculated from the time that the proxy receives LSR and DLSR fields of a RTCP Receiver Report message and the time that the proxy receives the message. By applying an exponentially weighted moving average functions, the smoothed values are derived for both. The estimated throughput obtained by Eq. (2.1) is regarded as the available bandwidth, which is taken into account in determining the quality of a block to retrieve and send.

To derive the TCP-friendly rate, the TFRC requires a client to send feedback messages at least once per RTT. It means that a client application has to issue RTCP Receiver Report messages at least once per RTT. According to the RTCP specifications, a sender can trigger feedback by sending an RTSP Sender Report to a receiver. However, widely available client applications such as used in the experiments in this thesis ignore this and issue RTCP Receiver Report messages every three to six seconds by their own timing. To verify the practicality and applicability of our proxy cache system, we used the client applications as they are, without any modification. As a result, we observed large variation in the reception rate as will be shown in Section 2.3.1. Problems inherent in public applications remains for future research.

2.2.4 Video Quality Adaptation

We employed a frame dropping filter as a quality adaptation mechanism. The frame dropping filter adapts the video quality to the desired level by discarding frames. The smoothness of video play-out deteriorates but it is simpler and faster than other filters such as low-pass and re-quantization [79]. Adopting layered or multiple-description coding is also helpful to treat the client-to-client heterogeneity. However, no commercially or freely available client application can decode and display a media stream with multiple layers or multiple descriptions.

We should take into account the interdependency of video frames in discarding frames. For example, discarding an I-VOP affects P-VOPs and B-VOPs that directly or indirectly

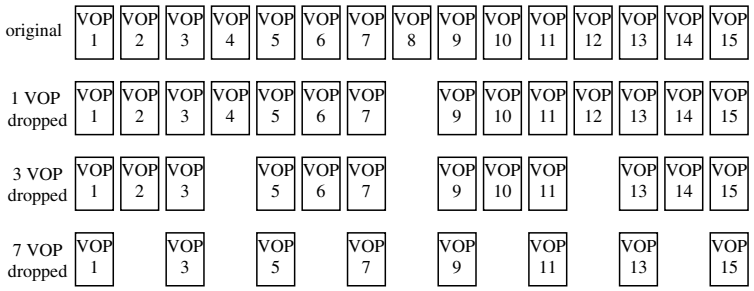


Figure 2.3: Video structure after frame dropping

refer to the I-VOP in coding/decoding processes. Thus, unlike other filters, we cannot do packet-by-packet or VOP-by-VOP adaptation. Therefore, in our proxy cache server, the frame dropping filter is applied to a series of VOPs of one second. The *Filter* first buffers, e.g. 15 VOPs in our system where the video frame rate is 15 fps. Then, the order for discarding is determined.

To keep the smoothness of video play-out preferably, we propose an algorithm to decide the frame dropping order. We first prepare a binary tree of VOPs and discard frames in a well-balanced manner. The VOP at the center of the group, i.e. the eighth VOP in the example, became the root of the tree and was given the lowest priority. Children of the eighth VOP were the fourth and twelfth VOPs and respectively became the second and third candidates for frame dropping. Figure 2.3 shows the resulting sequences of VOPs after frame dropping. As shown Fig. 2.3, this algorithm makes the number of VOPs among groups divided by gaps the same to have smooth video play-out. However, the order itself does not take into account VOP types. Then, considering inter-VOP relationships, we first discard B-VOPs from ones that have the lowest priority until the amount of video data fits the bandwidth. If discarding all B-VOPs is insufficient to attain the desired rate, we move to P-VOPs. Although we could further discard I-VOPs, they have been kept in the current implementation for the sake of smooth video play-out without long pauses.

Figure 2.4 shows bit rate variation of filtered video streams generated aiming at 200, 500, 800 kbps from the original VBR video stream whose average rate is 1000 kbps. Although our filtering algorithm is simple, resultant video rates are close to target rates with small fluctuations. We can replace ours with any other better algorithm as far as an off-the-shelf client application can decode a manipulated video stream.

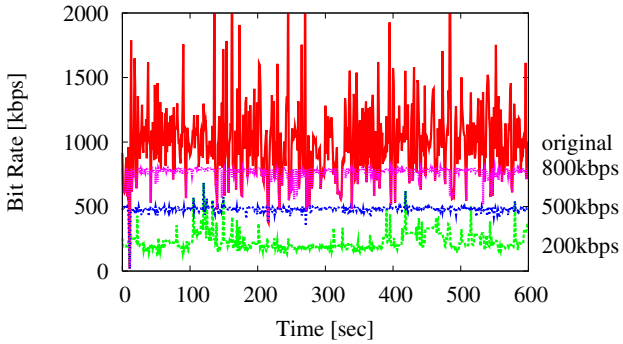


Figure 2.4: Adapted video rate by frame dropping filter

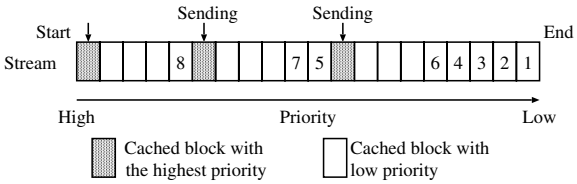


Figure 2.5: Priority of cached blocks

2.2.5 Block Prefetching

To reduce the possibility of cache misses and avoid the delay in obtaining missing blocks from a server, a proxy prefetches blocks that clients are going to require in the future. In a case of a cache hit, the *Cache Manager* examines the *Cache Table* in succeeding P blocks. Here, P is the size of a sliding window, called a prefetching window, which determines the range of examination for prefetching. As long as blocks are cached, the *Cache Manager* sequentially reads them out and sends them to the *RTP Sender*. If there exists any block which is not cached in succeeding P blocks, the *Cache Manager* prefetches the missing block by sending an RTSP PLAY message to the video server. The *Cache Manager* also prefetches succeeding blocks that are not cached.

2.2.6 Cache Replacement

When a proxy cache server retrieves a block and finds the cache is full, it discards one or more less important blocks to make room for the new block. With our cache replacement algorithm, first, the *Cache Manager* selects a video stream with the lowest priority from

cached streams using an LRU algorithm. It then assigns priorities to blocks in the selected stream using the following algorithm. Blocks being sent to a client have the highest priority. The block at the beginning of the stream is also assigned the highest priority to provide potential clients with a low-latency service. Among the others, those closer to the end of a longer succession of cached blocks are given lower priorities. Finally, blocks candidate for replacement are chosen one by one until sufficient capacity becomes available.

Figure 2.5 illustrates an example of victim selection. In this example, the block located at the end of the stream is in the longest succession. Therefore, the block is given the lowest priority and becomes the "1"st victim. Among successions of the same length, the one closer to the end of the stream has lower priority.

2.3 Experimental Evaluation

In this section, we show results of experiments on a prototype. In the experiments, we use 10 and 30 minutes long video streams by coding animation, scenery, action, fantasy, computer graphic, and sports movies using an MPEG-4 VBR coding algorithm at the coding rate of 1 Mbps. Video data of 320×240 pixels and 30 fps and audio data of 96 kbps are multiplexed into an MPEG-4 stream. The maximum and minimum bit rate are about 2 Mbps and 400 kbps, respectively. An example of rate variation is illustrated in Fig. 2.4 as "original". The size of the video stream is about 84 Mbytes for 10 minute stream and 248 Mbytes for 30 minute stream, respectively. A block corresponds to 300 VOPs, i.e. 10 seconds. Thus, the stream consists of 60 or 180 blocks. A video server always has the whole video blocks. A client watches a video from the beginning to the end without interactions such as rewinding, pausing, and fast-forwarding.

There have been proposals for proxy caching mechanisms for video streaming service [16-23]. However, they do not consider the client-to-client heterogeneity, lack the scalability and adaptability to rate and quality variations, or assume specially designed server/client applications which are not widely available. There is no suitable work or implementation to compare with our system. Therefore, we do not show comparison results with other research work in following sections.

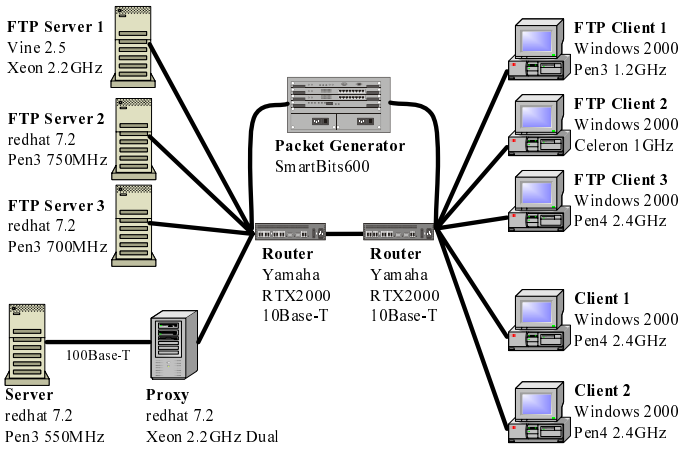
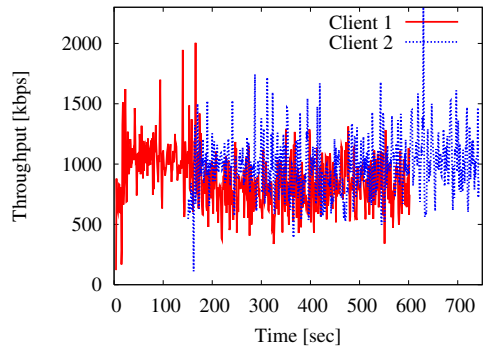


Figure 2.6: Configuration of experimental system to evaluate rate control

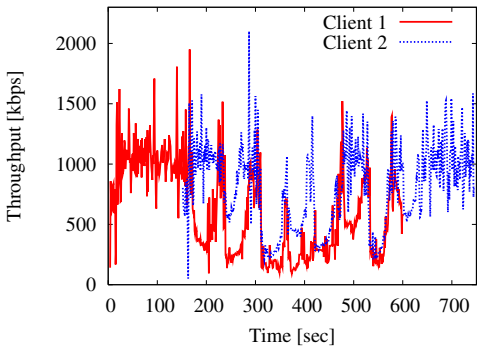
2.3.1 Rate Control with Video Quality Adaptation

Figure 2.6 illustrates the configuration for our experimental system to evaluate the availability of rate control with video quality adaptation. A proxy is directly connected to a video server. Two video clients are connected to the proxy through two routers. Video sessions compete for the bandwidth of the link between two routers with three FTP sessions and a UDP flow generated by a packet generator. The proxy has no blocks and a cache buffer capacity is limited to 50 Mbytes. The prefetching window size P is set to 5. Video client 1 issues an OPTIONS message at time zero, and video client 2 issues it at 150 seconds. Two clients watch the same video stream. FTP sessions start transferring files at 300 seconds and stop at 450 seconds. The packet generator always generates UDP packets at the rate of 8 Mbps. For purposes of comparison, we also conducted experiments using a proxy without the capability of quality adaptation, which is called the traditional system hereafter.

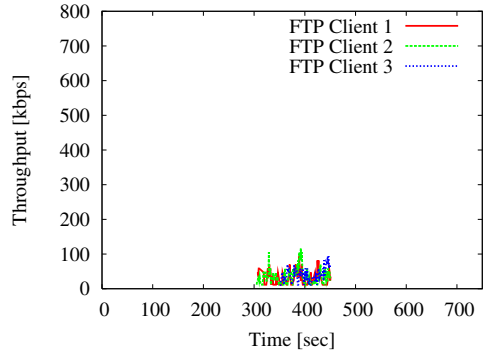
Figure 2.7 illustrates variations in reception rates observed at each client with *tcpdump*. As Fig. 2.7(b) shows, the reception rate changes in accordance with network conditions. During congestion, the average throughput of TCP sessions is 277 kbps with our system. On the contrary, since the traditional system keeps sending video traffic at the coding rate as shown in Fig. 2.7(a), TCP session are disturbed and, the attained throughput is only 37 kbps. As a result, the friendliness is 1.44 in our system and 23.1 in traditional system, where the friendliness is given by dividing the average throughput of video sessions by that of TCP.



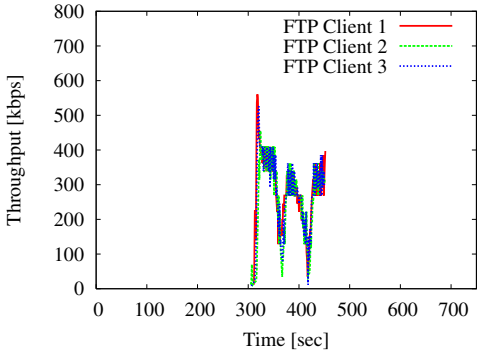
(a) Video reception rate with traditional system



(b) Video reception rate with quality adaptation

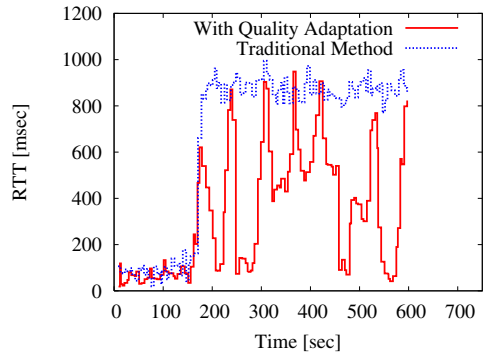


(c) FTP reception rate with traditional system

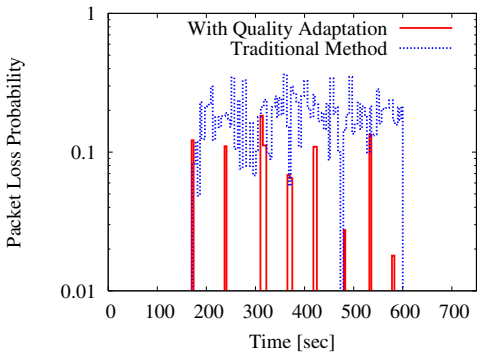


(d) FTP reception rate with quality adaptation

Figure 2.7: Reception rate at clients



(a) RTT



(b) Packet loss probability

Figure 2.8: RTT and packet loss probability at client 1

However, as observed in Fig. 2.7(b), there are large rate variations in video sessions. The average throughput of video sessions during the competitive period is higher than that of TCP sessions. TCP sessions are sensitive to congestion and they suppress the number of packets to inject into the network when they occasionally observe packet losses. Video sessions, on the other hand, do not notice sudden and instantaneous packet losses due to the long control intervals. The major reason for this is that the control interval of adaptation is three to six seconds due to the problem of the client application as described in Section 2.2.3. The interval is considerably longer than that of TCP, which reacts to network conditions in an order of RTT. By increasing the frequency that a client reports feedback information by modifying the client applications, such discrepancies are expected to be eliminated. Another reason is that the experimental system is relatively small. As a result, only a slight change during a session directly and greatly affects the other sessions. Then, synchronized behaviors are observed in Fig. 2.7(b) and 2.7(d).

Figure 2.8 shows RTT and packet loss probability calculated from information in RTCP Receiver Report messages. In the traditional system, the proxy persists in sending video data at the coding rate during congestion, and many packets are delayed or lost at routers. The packet delay may cause freezes at play-out due to underflow of play-out buffer. Furthermore, the client application abandons playing out a VOP which is seriously damaged by a packet loss. The influence of a packet loss propagates to the other VOPs when I-VOP or P-VOP is damaged. During the experiment, 9712 VOPs were played out with our system, but only 9133 VOPs were played out with the traditional system at client 1. Therefore the perceived video quality is higher and smoother with our system than with the traditional system owing to the intentional frame discarding, although the amount of received video data in the traditional system is larger than that in our system.

2.3.2 Effectiveness of the Caching Mechanism

Figure 2.9 illustrates the configuration for our experimental system to evaluate our proxy caching mechanism. A proxy is connected to a video server through a router. Three clients are directly connected to the proxy. In order to control the delay, *NISTNet* [80], a network emulator, is used at the router. The one-way delay between the video server and the proxy is set to 125 msec. Clients 1 through 3 issue an OPTIONS message at time 0, 350, and 700, respectively. Three clients watch the same video stream. The proxy has no block at first. We do not introduce rate control with quality adaptation at the proxy. For purposes

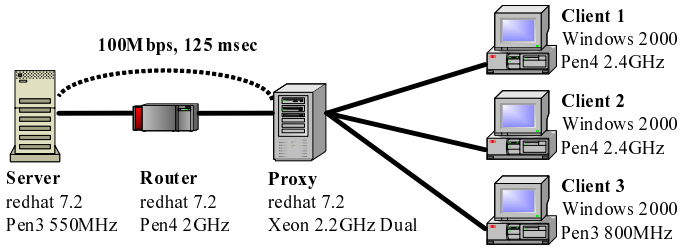
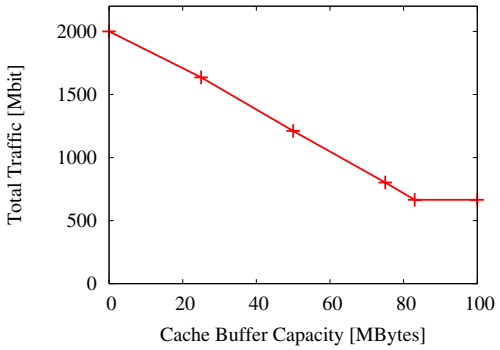
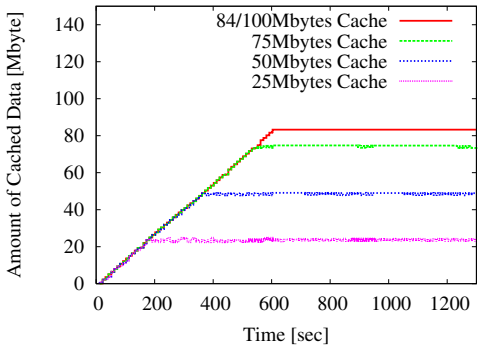


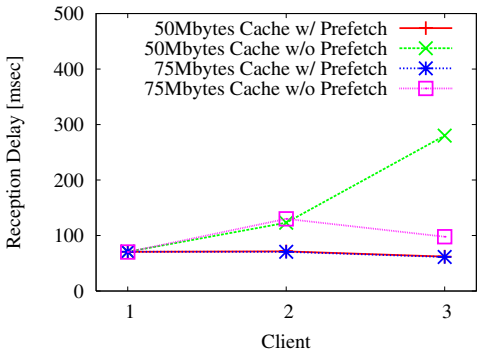
Figure 2.9: Configuration of experimental system to evaluate the caching mechanism



(a) Total traffic between server and proxy



(b) Amount of cached data



(c) reception delay

Figure 2.10: Evaluation of the caching mechanism

of comparisons, we also conducted experimental evaluations of cases where the proxy has no cache buffer, that is, when clients always received video blocks from the server.

Figure 2.10(a) shows the total amount of traffic between the video server and the proxy during experiments, and Fig. 2.10(b) shows the amount of cached data. Prefetching window size P is set to zero, i.e. no prefetching. In Fig. 2.10(a), 0 Mbyte of the cache buffer capacity means the proxy has no cache buffer. As the buffer capacity increases, the total amount of traffic between the server and the proxy decreases. When the buffer capacity exceeds 84 Mbytes, i.e. the size of the whole video stream, the total amount of traffic does not change any more. In addition, the amount of cached blocks is kept within the limitation of buffer capacity as Fig. 2.10(b) shows. Consequently, it is shown that the proxy can provide clients with blocks from its local cache buffer by replacing less important blocks with newly retrieved blocks.

We define the reception delay of client j, d_j, as follows,

$$d_j = \frac{1}{N}\sum_{i=1}^{N}(T_j(i) - T_j(1) - i/F), \tag{2.2}$$

where, N corresponds to the number of VOPs in a stream, and F corresponds to the frame rate. $T_j(i)$ is the time that client j receives the VOP i. Thus, the reception delay d_j is the sum of differences between the expected arrival time of a VOP and the actual arrival time at a client. Figure 2.10(c) shows the average of reception delay during a video session at each client while prefetching window P is set to 0 or 5. Since there is no cached block in the proxy at first, the reception delay of client 1 is the same whether the proxy conducts prefetching or not. However, for client 2 and 3, the reception delay without prefetching is larger than that with prefetching, since there is the delay in obtaining missing blocks from the server. Specifically, when the buffer capacity is 50 Mbytes, the reception delay on client 3 with a non-prefetching proxy is 280 msec. When client 3, the last client among three, joined the service, some parts of a video stream had been swept out from a cache buffer due to the limited capacity. As a result, the number of blocks missing in a cache buffer is larger than the other two clients. When a proxy does not have a capability of prefetching, it has to retrieve all missing blocks from a video server when they are requested by a client. It introduces delay. Consequently, the reception delay increases.

In this experiment, since we consider a small and underloaded network, the reception delay is small enough without the capability of prefetching. We expect that the delay

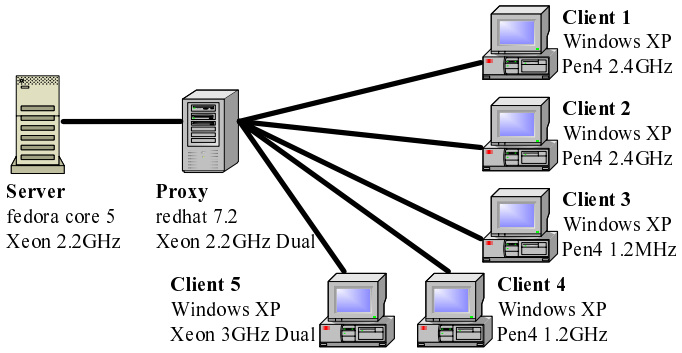
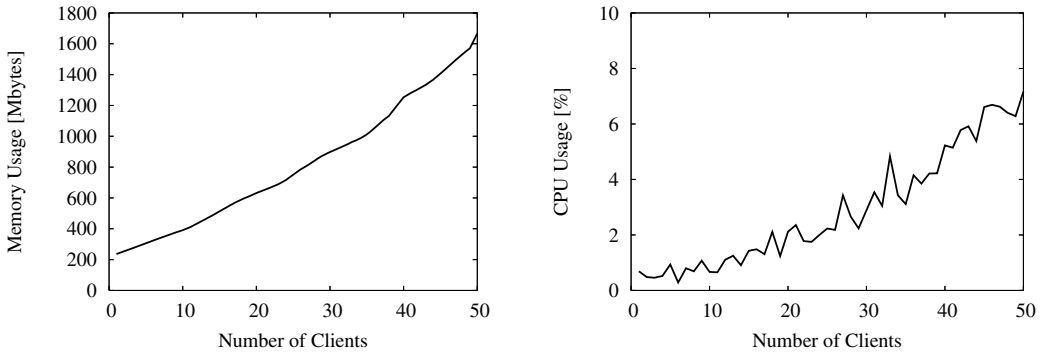


Figure 2.11: Configuration of experimental system to evaluate with many clients



(a) Memory usage (b) CPU usage

Figure 2.12: Average memory usage and CPU usage for the number of clients

exceeds the initial buffering of three seconds in a larger network. However, by introducing the prefetching mechanism and a larger value of P, user becomes free from annoying freezes.

2.3.3 Evaluation with Many Clients

Finally, we conducted experiments on a system with many clients. Figure 2.11 illustrates the configuration for the experiments. In our experimental configuration, a proxy with 200 MBytes cache is directly connected with a video server and five clients. 10 client applications are running on each client and request the same video stream of 30 minutes. The client applications issue an OPTIONS message at random and begin watching the movie. The proxy has no block at first. The load on the proxy in terms of the memory

and CPU usage is measured by using *vmstat* every one second, while increasing the number of client applications. Although not shown in figures, we verified the smoothness of video play-out on a monitor for a case of many clients.

Figure 2.12 shows the average memory usage and average CPU usage for the number of clients. As shown in Fig. 2.12(a), the memory usage is almost in proportional to the number of clients, because a process of proxy caching modules shown in Fig. 2.1 is invoked for each client on the proxy cache server. The CPU usage also linearly increases with the number of clients, but it is less than 8% for 50 clients as shown in Fig. 2.12(b).

In conclusion, although we confirmed that our system can offer heterogeneous services to more than 50 clients, we also observed that the load on the proxy cache server is proportional to the number of clients in the current implementation. By using the latest equipment and applications and optimizing of the system, we can expect that a proxy cache server can accommodate more clients. However, to have the higher scalability, we need to improve our system, whereas we adopted rather general approaches and mechanisms in the current implementation.

2.4 Conclusion

In this chapter, we proposed, designed, implemented, and evaluated a proxy caching system for MPEG-4 video streaming services. We employed off-the-shelf and common applications for server and client programs to verify the practicality of our proposed system. Through experiments, it was shown that our proxy caching system could dynamically adapt the quality of video streams to network conditions while providing users with a continuous and high-quality video streaming service. We also verified that our system could provide 50 clients with smooth video streaming.

Chapter 3

A Cooperative Proxy Caching Mechanism with Quality Adaptation for Video Streaming Services

3.1 Introduction

In the previous chapter, we proposed, designed, implemented, and evaluated a proxy caching system for MPEG-4 video streaming services using off-the-shelf and prevailing products. However, the system was intended for only a single proxy. A proxy always retrieves blocks from a video server at cache miss although there are other proxies having low-delay and broadband links to the proxy. By considering cooperation among proxies, further effective and high-quality video streaming service can be provided.

In this chapter, we extend our previous work by considering cooperation among proxies. The new mechanism consists of three parts: block provisioning, block prefetching, and cache replacement. The main benefits of our proxy cooperation mechanism include reducing the perceived network latency and achieving a higher degree of content availability by cooperative caching. Through simulation experiments, it is shown that our proposed mechanism can provide users with low-delay and high-quality video streaming services. In addition, to verify the practicality and adaptability of our proposal to existing video streaming services,

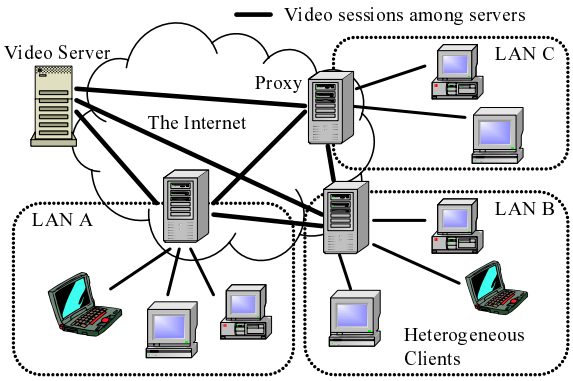


Figure 3.1: Cooperative video streaming system

we implement our cooperative proxy caching mechanism on a real system for MPEG-4 video streaming services extending our implemented system described in Chapter 2. We employ off-the-shelf and common applications for server and client systems. Our implemented system is designed to conform to the standard protocols. Through experimental evaluations, it is shown that our proxy caching system can provide users with a continuous video distribution under dynamically changing network conditions.

The rest of this chapter is organized as follows. In Section 3.2, we propose the cooperative proxy caching mechanism for video streaming services. Next, in Section 3.3, we evaluate our proposed mechanism through simulation experiments. In Section 3.4, we describe the implementation of the proposed mechanism on a real system and conduct several experiments. Finally, we conclude this chapter in Section 3.5.

3.2 A Cooperative Proxy Caching Mechanism with Video Quality Adaptation

In this section, we describe our cooperative proxy caching mechanism. The video streaming system we consider in this chapter is illustrated in Fig. 3.1. Our system consists of a single video server, cooperative proxies, and heterogeneous clients. Clients are heterogeneous in regard to their available bandwidth, propagation delays, end-system performance, and user preferences on the video quality.

A video stream is divided into L blocks, each consisting of B frames and assume a

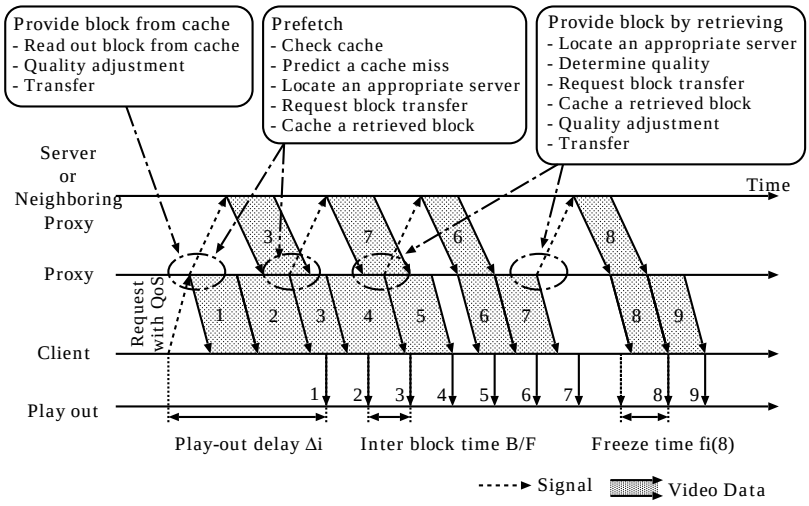


Figure 3.2: Basic behavior of our cooperative proxy caching system

constant frame rate of F. A proxy retrieves video blocks from the distant video server or neighboring proxies on behalf of clients, deposits them in its local cache buffer, and adapts the quality of the blocks to the user's demands. In addition, proxies communicate with each other and exchange blocks over a video session established among them, and maintain information of cached blocks at the other servers. We should note here that, to reduce the load on the network and servers, there is only a single session established between any pair of servers regardless of the number of clients.

In the following sections, we describe the involved mechanisms for the video streaming system to provide users with low-delay and high-quality services under a heterogeneous environment.

3.2.1 Overview of the Mechanism

Figure 3.2 illustrates how servers and client communicate with each other. The numbers in the figure correspond to block identifiers, which are in the order of their playout. The video streaming service is initiated by a request message issued by a client to a designated proxy. The message contains information about preferable (upper-constraint) and tolerable (lower-constraint) levels of video quality which are determined in accordance with the available bandwidth, end-system performance, and user preferences. A client is allowed to dynamically change QoS requirements during a video session to reflect changes of those

constraints.

On receiving the first request message, the proxy begins to provide a requested video stream to the client in a block-by-block fashion. The proxy adopts the fastest way that can provide the client with a block of high level of quality. The proxy can 1) read out and send a cached block, 2) use a block being received, 3) wait for the preceding request for the same block to be served, or 4) newly retrieve a block from another server. When the proxy finishes sending out the block, it moves to the next block and repeats the same procedure.

While providing clients with video blocks, the proxy predicts and prepares for a future potential cache miss by prefetching a block from other servers. A prefetching request is processed without disturbing normal block retrievals. Details of the prefetching mechanism will be given in Section 3.2.3.

To handle unexpected block transmission delays, client i first defers playing out a received video block for a predetermined waiting time Δ_i as shown in Fig. 3.2. Then, it begins to decode and display received blocks one after another, at regular intervals of B/F. In some cases such as a cache miss, a block does not arrive in time and the client is forced to pause. The time required for client i to wait until the arrival of block j is called *freeze time* and denoted as $f_i(j) \geq 0$ in this thesis

3.2.2 Block Provisioning

In providing a client with a block, a proxy chooses the best way among the four described in Section 3.2.1 in accordance with the offerable quality and the block transfer delay. For this purpose, servers communicate with each other and maintain the up-to-date information on other servers, such as the quality of offerable blocks, round-trip time, and available bandwidth in two tables. Information on locally cached blocks is maintained in the cache table, while the remote table is for information on cached blocks at other servers. To predict the transfer time as accurately as possible, a proxy is assumed to be able to estimate the block size, propagation delay, and throughput. Information related to the network condition among a proxy and its clients is also required.

Assume that proxy k is trying to provide client i with block j at time t. The quality of block j must be higher than $q_i(j)$ and as high as $Q_i(j)$, which are determined by QoS requirements specified by the latest request message. The deadline $T_i(j)$ that client i should

finish receiving block j is determined as

$$T_i(j) = T_i + \Delta_i + (j-1)\frac{B}{F} - \delta_i + D_i(j-1), \tag{3.1}$$

where T_i indicates the instant when client i issues the first request message. δ_i is introduced to absorb unexpected delay jitters and estimation errors. $D_i(j-1) = \sum_{l=1}^{j-1} f_i(l)$ is the accumulated freeze time. We consider four cases of providing a block to a client as follows. For each case, a proxy estimates the offerable quality and takes the best choice.

1) Successful Cache Hit

The first case is that the desired block j already exist in the cache buffer of proxy k. The offerable quality $c_{k,i}^{PC}(j)$ of block j to client i by using block j cached at the proxy k's buffer is derived as

$$c_{k,i}^{PC}(j) = \min(q_k(j), \overline{c}_{k,i}^{PC}(j)), \tag{3.2}$$

where

$$\overline{c}_{k,i}^{PC}(j) = \max(q | t + d_{k,i}^{PC}(t) + \frac{a_j(q)}{r_{k,i}^{PC}(t)} \leq T_i(j)). \tag{3.3}$$

$q_k(j)$ stands for the quality of block j cached at the proxy k's buffer. $a_j(q)$ is a function indicating the size of block j of quality q and is defined as 0, if $j = 0$ or $q = 0$. This function depends on the employed codec and we will provide an example later in Section 3.3. $d_{k,i}^{PC}(t)$ and $r_{k,i}^{PC}(t)$ are estimates of one-way propagation delay and available bandwidth from proxy k to client i at t, respectively. Thus, $d_{k,i}^{PC}(t) + \frac{a_j(q)}{r_{k,i}^{PC}(t)}$ provides the estimated time required for client i to receive the whole of block j of quality q via a video session whose available bandwidth is $r_{k,i}^{PC}(t)$ and propagation delay is $d_{k,i}^{PC}(t)$.

2) Cache Miss and Block Download in Progress

The second case is that the block j is not available in the cache at proxy k and it is currently being retrieved from another server s. The quality offerable is derived as

$$b_{k,i}^{PC}(j) = \max_{\forall s, s \neq k}(\min(q_{s,k}^{SP}, \overline{b}_{k,i}^{PC}(j))), \tag{3.4}$$

where

$$\overline{b}_{k,i}^{PC}(j) = \max(q|t + d_{k,i}^{PC}(t) \ + \max(\frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}, \frac{a_j(q)}{r_{k,i}^{PC}(t)}) \leq Ti(j)). \tag{3.5}$$

In the above equations, s indicates the server from which the proxy is receiving block j. $q_{s,k}^{SP}(t)$ is the quality of block being received at t. $a_{s,k}^{SP}(t)$ stands for the amount that has already been received. $d_{s,k}^{SP}(t)$ and $r_{s,k}^{SP}(t)$ correspond to estimates of one-way propagation delay and available bandwidth from server s to proxy k at t.

3) Cache Miss and Waiting for Pending Block

The third case is that the block j is not available in the cache at proxy k, but it already submitted a request for j to another server s. A proxy keeps track of requests that it sent out for block retrieval. $\mathcal{W}_{s,k}^{SP}(t) = \{W_1, W_2, \ldots, W_w\}$ is a list of requests that had been sent from proxy k to server s before t. The quality $q_{s,k}^{w}(n)$ of block W_n is also maintained in a list. A pair of the block number and the quality is added to the lists when the proxy sends a request for block retrieval and is removed from the lists when the proxy begins receiving the block. The offerable quality $w_{k,i}^{PC}(j)$ for the m-th request, which is block j, i.e., $W_m = j$, can be estimated as

$$w_{k,i}^{PC}(j) = \max_{\forall s, s \neq k}(\min(q_{s,k}^{w}(m), \overline{w}_{k,i}^{PC}(j))), \tag{3.6}$$

where

$$\begin{aligned} \overline{w}_{k,i}^{PC} \quad = \quad & \max(q|t + 2d_{s,k}^{SP}(t) + d_{k,i}^{PC}(t) + \frac{\sum_{n=1}^{m-1} a_{W_n}(q_{s,k}^{w}(n)) + a_{p_{s,k}(t)}(q_{s,k}^{p}(t))}{r_{s,k}^{SP}(t)} \\ & + \max(\frac{a_j(q_{s,k}^{w}(m))}{r_{s,k}^{SP}(t)}) \leq T_i(j)). \end{aligned} \tag{3.7}$$

$p_{s,k}(t)$ indicates the block number to be prefetched from server s and $q_{s,k}^{p}(t)$ is its quality. If no prefetching request is waiting for server s, both $p_{s,k}(t)$ and $q_{s,k}^{p}(t)$ are zero. In our system, only one prefetching request is permitted by a server per proxy and prefetching is preempted by normal block retrievals. Details of the prefetching mechanism will be given in Section 3.2.3. Equation (3.7) considers the worst case when no block is under transmission and the preceding requests from 1 to $m-1$ and the prefetching request will be served prior

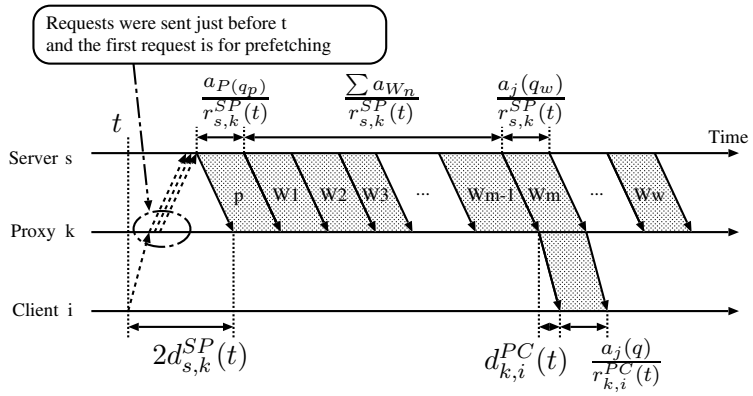


Figure 3.3: Worst-case delay due to waiting for the preceding request

to block W_m, as illustrated in Fig. 3.3. If the proxy is receiving a block at t, $\overline{w}_{k,i}^{PC}$ is derived using the following equations.

$$\begin{aligned}\overline{w}_{k,i}^{PC} &= \max(q|t+\max(2d_{s,k}^{SP}(t), \frac{a_j(q_{s,k}^{SP}(t))-a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}) \\ &\quad +d_{k,i}^{PC}(t)+\frac{S_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}+\max(\frac{a_j(q_{s,k}^{w}(m))}{r_{s,k}^{SP}(t)}, \frac{a_j(q)}{r_{k,i}^{PC}(t)}) \le T_i(j)), \end{aligned} \tag{3.8}$$

where

$$S_{s,k}^{SP}(t) = \begin{cases} \sum_{n=1}^{m-1} aw_n(q_{s,k}^{w}(n)) + a_{p_{s,k}(t)}(q_{s,k}^{p}(t)), & \text{if } 2d_{s,k}^{SP}(t) > \frac{a_j(q_{s,k}^{SP}(t))-a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} \\ \sum_{n=1}^{m-1} aw_n(q_{s,k}^{w}(n)), & \text{otherwise} \end{cases}. \tag{3.9}$$

4) Cache Miss and New Block Request

The final case is that the block j is not available in the cache at proxy k and a new request must be sent to anther server s. The offerable quality $s_{k,i}^{PC}(j)$ is derived by the following equations when there is no block under transmission,

$$s_{k,i}^{PC} = \max_{\forall s, s\neq k}(\min(q_s^r(j), \overline{s}_{k,i}^{PC}(j))), \tag{3.10}$$

where

$$\overline{s}_{k,i}^{PC} = \max(q|t + 2d_{s,k}^{SP}(t) + d_{k,i}^{PC}(t) + \frac{\sum_{n=1}^{w} aw_n(q_{s,k}^{w}(n)) + a_{p_{s,k}(t)}(q_{s,k}^{p}(t))}{r_{s,k}^{SP}(t)} + \frac{a_j(q)}{\min(r_{s,k}^{SP}(t), r_{k,i}^{PC}(t))} \leq T_i(j)). \quad (3.11)$$

Here, $q_s^r(j)$ corresponds to the quality of block j cached at server s. On the contrary, if proxy k is receiving a block from server s,

$$\overline{s}_{k,i}^{PC} = \max(q|t + \max(2d_{s,k}^{SP}(t), \frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)}) + d_{k,i}^{PC}(t) + \frac{U_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} + \frac{a_j(q)}{\min(r_{s,k}^{SP}(t), r_{k,i}^{PC}(t))} \leq T_i(j)), \quad (3.12)$$

where

$$U_{s,k}^{SP}(t) = \begin{cases} \sum_{n=1}^{w} aw_n(q_{s,k}^{w}(n)) + a_{p_{s,k}(t)}(q_{s,k}^{p}(t)), & \text{if } 2d_{s,k}^{SP}(t) > \frac{a_j(q_{s,k}^{SP}(t)) - a_{s,k}^{SP}(t)}{r_{s,k}^{SP}(t)} \\ \sum_{n=1}^{w} aw_n(q_{s,k}^{w}(n)), & \text{otherwise} \end{cases}. \quad (3.13)$$

The fastest and best way is chosen among the four as far as the offerable quality is above $q_i(j)$ and below $Q_i(j)$. If none of $c_{k,i}^{PC}(j)$, $b_{k,i}^{PC}(j)$, $w_{k,i}^{PC}(j)$, and $s_{k,i}^{PC}(j)$ can satisfy request $q_i(j)$, the proxy chooses the fastest way.

3.2.3 Block Prefetching

While supplying client i with block j, a proxy investigates its cache table for blocks $j+1$ to $j+P$ to find a potential cache miss. The parameter P determines the size of the prefetching window. When the proxy finds the block with quality lower than $q_i(j)$ and there is no request waiting to be served, it attempts to prefetch the block of higher quality from another server. If there are two or more unsatisfactory blocks, the one closest to block j is chosen.

Prefetch requests are treated at a server in a different way from requests for retrieving cache-missed blocks. The video server and proxies maintain a pair of prioritized queues per video session. The queue for usual block retrieval is given a higher priority and requests are

handled in a first-come-first-served discipline. On the other hand, the queue for prefetching has a limited length of 1 and a waiting request is always overwritten by a new one. A prefetch request in the queue is served only when there is no request in the high-priority queue. A prefetch request is considered obsolete and removed when a server receives a request for the same block with higher quality.

A proxy decides to prefetch a block if reception of the block is expected to be completed in time. The expected time $t^p_{s,k}(t)$ when the proxy finishes prefetching is derived as

$$t^p_{s,k}(t) = t + 2d^{SP}_{s,k}(t) + \frac{a_{p_{s,k}(t)}(q^p_{s,k}(t))}{r^{SP}_{s,k}}, \tag{3.14}$$

where $p_{s,k}(t)$ and $q^p_{s,k}(t)$ stand for the block number and the requested quality, respectively. This means that the proxy tries prefetching only when no preceding request is waiting to be served. If the derived time $t^p_{s,k}(t)$ is earlier than

$$T^p_i(p_{s,k}(t)) \quad = \quad T_i + \Delta_i + (p_{s,k}(t) - 1)\frac{B}{F} - \delta_i + D_i(j-1) - d^{PC}_{k,i}(t), \tag{3.15}$$

the proxy sends a request to server s.

Quality $q^p_{s,k}(t)$ is determined on the basis of the QoS requirement as $\beta Q_i(j)$ where $0 < \beta \leq 1$. If we set β to a small value, we can expect to prefetch a large number of blocks, but their quality becomes low. On the other hand, with a large β, there is little chance to successfully prefetch blocks in time, but a high-quality video stream can be provided with prefetched blocks. Information about a prefetching request is kept at a requesting proxy as a pair of block number $p_{s,k}(t)$ and quality $q^p_{s,k}(t)$ and is overwritten by a new prefetch and canceled when a block reception begins or a normal block retrieval is requested for the same block of higher quality.

3.2.4 Cache Replacement

When the available space of a cache buffer becomes insufficient to deposit a newly received block, a proxy first makes a list of blocks to be discarded. Those blocks which are located in prefetching windows are likely to be used earlier, and thus they will not be discarded. The first P blocks of a video stream are also considered important to suppress the initial delay. The rest of blocks are all candidates for replacement. The block m closest to the end of the longest run, i.e., a succession of non-prioritized blocks, becomes the first candidate. If the

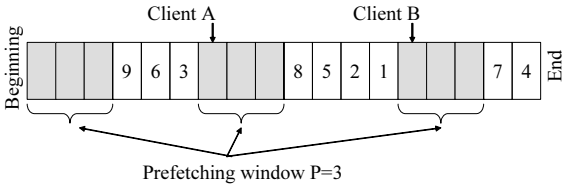


Figure 3.4: Sample order of block replacement

m-th block is cached, a proxy first tries degrading its quality to shrink the block size. The quality of the candidate should be larger than $\max_{1 \le j \le m-1} \max_{i \in S_{m-1}} Q_i(j)$ to prepare for future requests. Here, S_{m-1} is a set of clients which is watching any of blocks 1 through $m-1$. If the quality degradation is still insufficient, the proxy discards the candidate and moves to the next candidate at the end of the longest run. When all candidates, i.e., non-prioritized blocks, are dropped, but there is not enough room yet, the proxy gives up storing the new block. Figure 3.4 illustrates an example of an order of candidates. In the figure, the length of a video stream is 18 blocks and the prefetching window is $P = 3$ blocks. The proxy is serving clients A and B with a block 7 and 14, respectively. Then, those blocks from 1 to 3, from 7 to 9, and from 14 to 16 are not to be degraded or discarded. Among the others the first victim is block 13, since the block is located at the end of the longest run.

3.3 Simulation Experiments

In this section, we discuss the performance of our proposal with results we obtained through simulation experiments. The network we employed in the simulation is shown in Fig. 3.5. A video server is behind a wide-area network or the Internet. It communicates with five proxies through 10 Mbps sessions which are established over long-haul links with propagation delays 200 msec. Proxies are located at the boundary of an ISP network and are connected with each other via 20 Mbps sessions which are established over intra-network broadband links of 50 msec delay. A proxy establishes a 8 Mbps fixed-bandwidth session with each of its clients. The one-way propagation delay of each session is 10 msec. Although the available bandwidth and one-way delay fluctuate greatly under realistic conditions, we have employed static values so that we can clearly observe the basic behavior of our mechanisms. A client starts requesting video blocks at randomly determined times. The inter-arrival time of

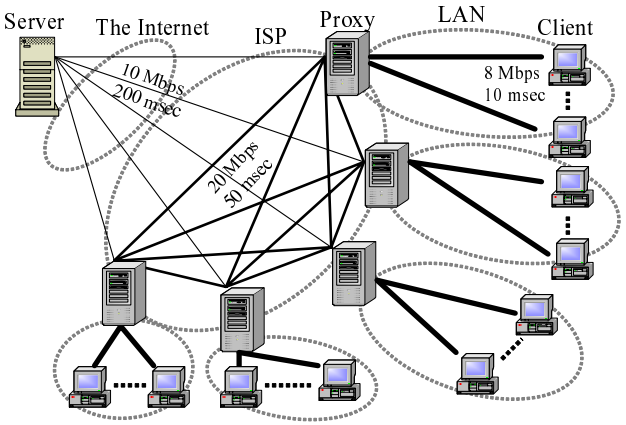


Figure 3.5: Configuration of simulation experiments

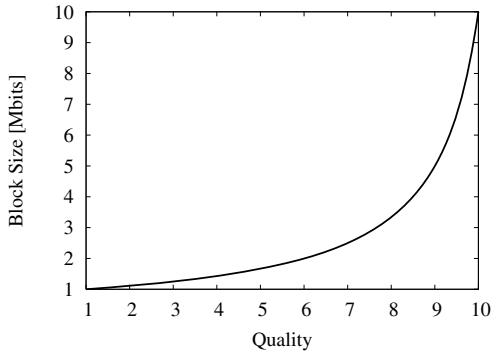


Figure 3.6: Relationship between quality and block size

the video requests issued by clients in each LAN follows an exponential distribution with average τ sec.

The video stream has a duration of 90 min and is played back at 30 fps. The stream is divided into 1 sec blocks, that is, $L = 5400$ and $B = 30$. There are ten levels of quality from $q_{\min} = 1$ to $q_{\max} = 10$, and they are mapped to block sizes from $a_{\min} = 1$ to $a_{\max} = 10$ Mbits. The relationship between quality and block size is usually described as convex downward [81]. In this thesis, we use $\forall j \; a_j(q) = \frac{a_{\max} q_{\min}}{q_{\max} + q_{\min} - q} \frac{B}{F}$ Mbits as shown in Fig. 3.6. Thus, the whole video stream amounts to 675 MB to 6.75 GB depending on the quality. Each proxy is equipped with a limited cache buffer. Initially, all cache buffers are empty and only the video server has all blocks of the highest quality. The prefetching window size

P is set at 10 blocks.

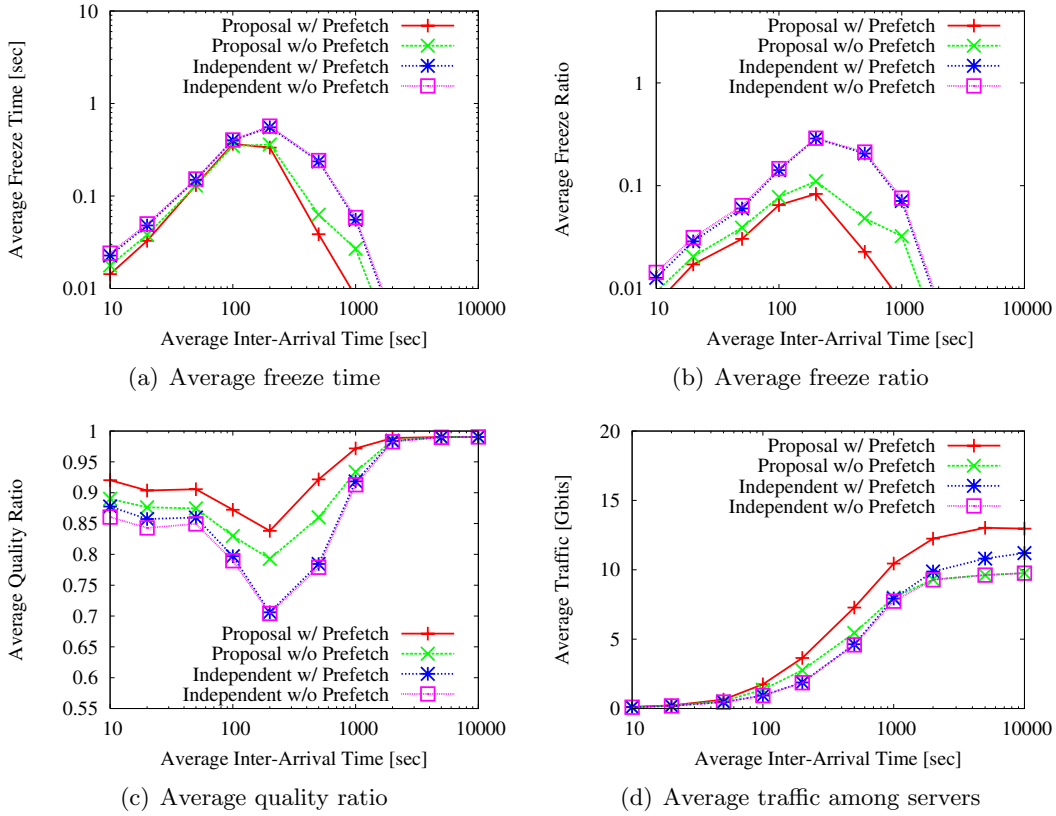
On client side, the buffering times Δ_i and δ_i to absorb delay jitters and prediction errors are set at 4 and 2 sec, respectively. The tolerable video quality $q_i(j)$ is fixed at 1. However, the preferable quality $Q_i(j)$ varies from 1 to 10, is initially set to 5, i.e., $Q_i(1) = 5$. Then, a client randomly determines the quality requirement on a block-by-block basis. In our experiments, the quality level $Q_i(j)$ is increased or decreased with a probability of 5% after each block is sent. This is introduced as a way of imitating the dynamic changes in quality requirements according to system conditions and the user preferences. The ratio of the quality of prefetched blocks to that of requests is determined as $\beta = 1.0$.

For comparison purposes, we conduct simulations of a system with four different schemes. One is referred as "independent w/o prefetch". In this approach proxies always retrieve the missing or unsatisfactory block from the originating video server without prefetching. "independent w/ prefetch" corresponds to the case where independent proxies are coupled with the prefetching mechanism. The schemes "proposal w/o prefetch" and "proposal w/ prefetch" indicate the corresponding cases where the proxies cooperate.

We compare the performance in terms of the average freeze time, the average freeze ratio, the average quality ratio, and the average traffic among servers per user. The average freeze time is derived as $\sum_{i=1}^{n} \sum_{j=1}^{L} f_i(j)/Ln$ where n is the number of clients. The average freeze ratio is defined as the ratio of number of freezes to the number of all blocks L per user. The quality ratio is defined as the ratio of quality of provided block to the requested preferable quality $Q_i(j)$. Simulations finish after 100τ sec and all results are averaged over 5 simulation runs.

First, we evaluate the effects of the inter-arrival time τ. The cache buffer size is fixed at 2 GB. Figure 3.7 shows the average freeze time, the average freeze ratio, the average quality ratio, and average amount of traffic among servers against the average inter-arrival time of clients. As shown in Fig. 3.7, our proposed mechanism can provide users with video distribution of lower delay and fewer freezes, achieving the same or higher quality ratio at the cost of a slight increase in traffic due to inter-proxy communication. Furthermore, the prefetching mechanism contributes to achieving low-delay and high-quality video distribution.

For all schemes, if the inter-arrival time is less than 200 sec, the average freeze time and the average freeze ratio increase with the inter-arrival time and the quality ratio decreases. This is because shorter inter-arrival times can achieve a higher probability of utilizing cached



(a) Average freeze time

(b) Average freeze ratio

(c) Average quality ratio

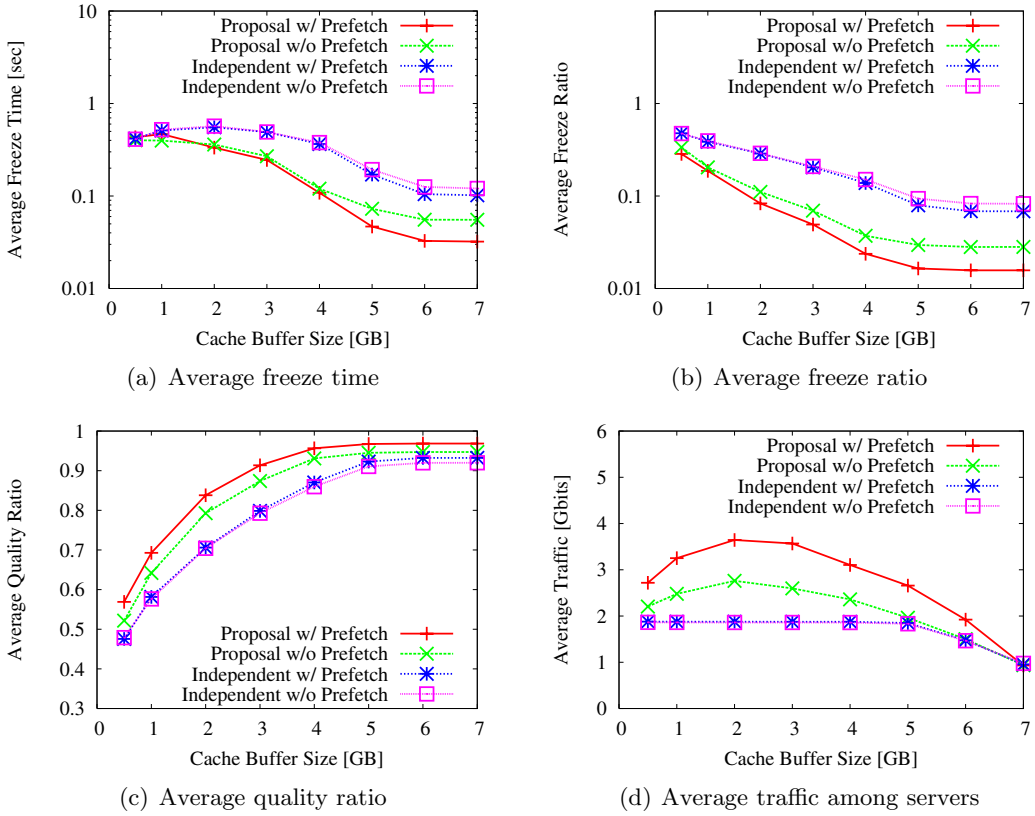
(d) Average traffic among servers

Figure 3.7: Simulation evaluations against average inter-arrival time τ

blocks and lower probability of retrieving new blocks from other servers. On the other hand, if the inter-arrival time is larger than 200 sec, as the inter-arrival time increases, the average freeze time and the average freeze ratio decrease and the quality ratio increases. This is because a proxy can use more bandwidth for each client, if the number of clients is small as shown in Fig. 3.7(d).

Next, we evaluate the effects of the cache buffer size. The average inter-arrival time τ is fixed at 200 sec. Figure 3.8 shows the average freeze time, the average freeze ratio, the average quality ratio, and the average amount of traffic among servers against cache buffer size. As shown in the figures, our proposed mechanism can provide users with video distribution of lower delay and fewer freezes, achieving higher quality ratio.

For all schemes, the average freeze time and the average freeze ratio decrease, the average quality ratio increases, and the average traffic among servers decreases with the



(a) Average freeze time

(b) Average freeze ratio

(c) Average quality ratio

(d) Average traffic among servers

Figure 3.8: Simulation evaluations against cache buffer size

cache buffer size as shown in Fig. 3.8. This is because a larger cache buffer size can achieve higher probability of cache hit and that of finding blocks in other proxies' cache buffers. However, in this experiment, increasing the cache buffer size more than 6 GB has no impact on the average freeze time, the average freeze ratio and the average quality ratio.

3.4 Implementation and Experimental Evaluation

In this section, we describe our implementation of proposed mechanism on a real system, and evaluate our mechanism through practical experiments. The system we implemented is based on our system described in Chapter 2, and video streaming is controlled through RTSP/TCP sessions. Each of the video and audio streams is transferred over a dedicated RTP/UDP session and the condition of streaming is monitored over RTCP/UDP sessions.

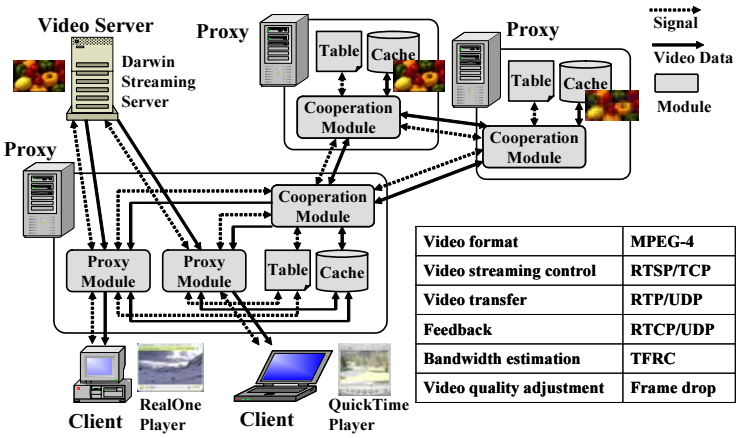


Figure 3.9: Overview of the implemented system

A video stream is coded using MPEG-4, and it is compliant with ISMA 1.0 [34]. In this chapter, we use the *Darwin Streaming Server* as a server application, and *RealOne Player* and *QuickTime Player* as client applications. However, other server or client applications being compliant with the standard can be incorporated with no or only small modification.

3.4.1 Overview of the Implemented System

Figure 3.9 illustrates the modules that constitute our video streaming system. Each dotted arrow and solid arrow corresponds to signal and data flow, respectively. A *Proxy Module* is generated for each client and provides a client with video blocks. A proxy has a *Cache* to deposit video data, and maintains the cache table and the remote table in the *Table*. We introduce a *Cooperation Module* for each proxy to communicate with neighboring proxies.

For its simplicity and speed, these modules have a frame dropping filter to adapt the quality of video. It adjusts the video quality to the desired level by discarding frames in a well-balanced way. To know the available bandwidths among the server, the proxies, and clients, they have the capability to estimate TCP-friendly [35] rates. Proxies estimate the throughput of a TCP session sharing the same path using control information obtained by exchanging RTCP messages.

In our implemented system, each block corresponds to a sequence of VOPs (Video Object Planes) of an MPEG-4 stream. A block consists of a video block and an audio block, and they are separately stored in the *Cache*. We empirically use $B = 300$ VOPs as

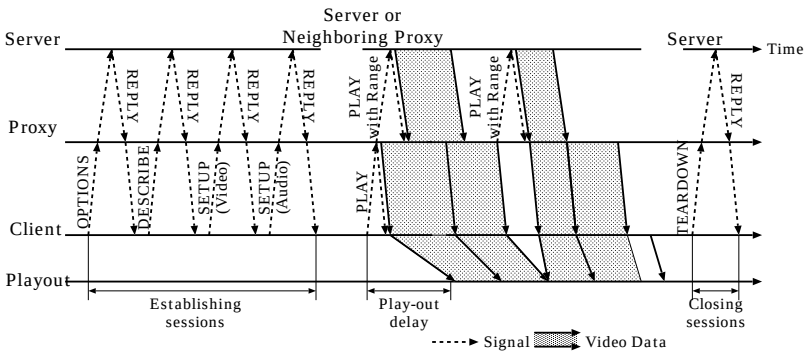


Figure 3.10: Basic behavior of the implemented system

the block size in our implementation. Since our MPEG-4 video stream is coded at $F = 30$ frames per second, a single block corresponds to 10 sec. We use the video coding rate to indicate its quality, and the Range and Bandwidth field of an RTSP PLAY message to specify the block and its quality.

The basic behavior of our system is illustrated in Fig. 3.10. First, a client establishes connections for audio and video streams with a proxy by sending a series of RTSP OPTIONS, DESCRIBE, and SETUP messages. These RTSP messages are received by the *Proxy Module* and relayed to the video server. Thus, connections between the video server and the proxy are also established at this stage. On receiving a SETUP REPLY message, the client requests delivery of the video stream by sending an RTSP PLAY message. Here, since the used client applications cannot declare an acceptable range of video quality levels, they are considered ready to receive and perceive a video stream at any quality.

The *Proxy Module* adopts the fastest way so that it can provide a client with a block of higher level of quality. When the *Proxy Module* provides a cached block, it reads it from *Cache* and sends it to the client. The quality of the video block is adjusted if necessary. When the *Proxy Module* retrieves a block from a neighboring proxy, it sends a request to the *Cooperation Module*. The *Cooperation Module* sends an RTSP PLAY message to the proxy, retrieves the block, and relays the block to the *Proxy Module*. When the reception is completed, the *Proxy Module* deposits the block in the *Cache*. If there is not enough room to store the newly retrieved block, the *Proxy Module* replaces the new block with less important blocks in the cache buffer. When the *Proxy Module* retrieves the block from the video server, it sends an RTSP PLAY message to the video server.

A client receives blocks from a proxy and first deposits them in the so-called play-out buffer. Then, it gradually reads blocks out from the buffer and plays them. When a proxy receives an RTSP TEARDOWN message from a client, the proxy relays the message to the video server, and closes the sessions.

3.4.2 Information Sharing Among Proxies

In order to maintain a remote table, a proxy issues an RTSP GET_PARAMETER message to other proxies when the end of the prefetching window of any client reaches an entry which is zero, i.e., an uncached block. An RTSP GET_PARAMETER message includes a list of blocks required in the near future. Blocks in the list are those which are currently requested by clients and its subsequent I blocks. These I blocks from the beginning of the stream are also listed in the message to prepare for new clients that will request the stream in the future. In addition, we also introduce a timer to force a refreshing of the remote table. The timer expires every $(I - P - 1)B/F$ and a proxy sends an RTSP GET_PARAMETER message to other proxies.

On receiving an RTSP GET_PARAMETER message, a proxy first examines its cache table about blocks listed in the message. It then returns an RTSP REPLY message which contains the list of pairs of a cached block and its quality.

3.4.3 Cooperative Proxy Caching

In order to run an implementation of our cooperative proxy caching mechanism as described in Section 3.2 on an actual system, we made the following small modifications.

Block Provisioning

A proxy adopts the fastest way that can provide a client with a block of higher quality in time. Since a server sends a video block frame-by-frame at the frame rate of a video stream in our implemented system, the value $a_j(q)/r^{SP}_{s,k'}(t)$ in the equations in Section 3.2.2 is replaced with B/F for the originating video server k'.

Block Prefetching

In our implemented system, a proxy sends an RTSP PLAY message with an additional field to prefetch a block from another proxy, that is the Prefetch field. Since the video server

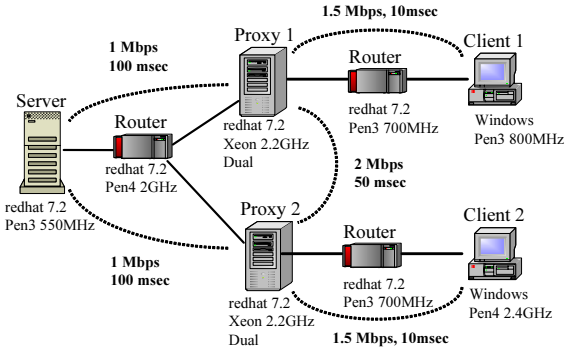


Figure 3.11: Configuration of experimental system

Cache table on proxy 1 (t=0)

block	bit rate [kbps]
b1-b3	1000
b4-b6	700
b7-b10	0
b11-b20	1000

Cache table on proxy 2 (t=0)

block	bit rate [kbps]
b1-b6	1000
b7-b20	0

Cache table on proxy 1 (t=200)

block	bit rate [kbps]
b1-b20	1000

Cache table on proxy 2 (t=400)

block	bit rate [kbps]
b1-b19	1000
b20	276

Figure 3.12: Cache tables

cannot process this new field, a proxy only sends a prefetching request to other proxies and not to the video server.

Cache Replacement

Since the client application does not declare its desired level of quality, a proxy only discards a cached block once it is chosen as a candidate.

3.4.4 Experimental Configuration

The effectiveness of the caching mechanism such as block prefetching and cache replacement as well as the scalability against the number of clients in our implemented system have been already verified in the case of a single proxy as described in Chapter 2. In this chapter, we

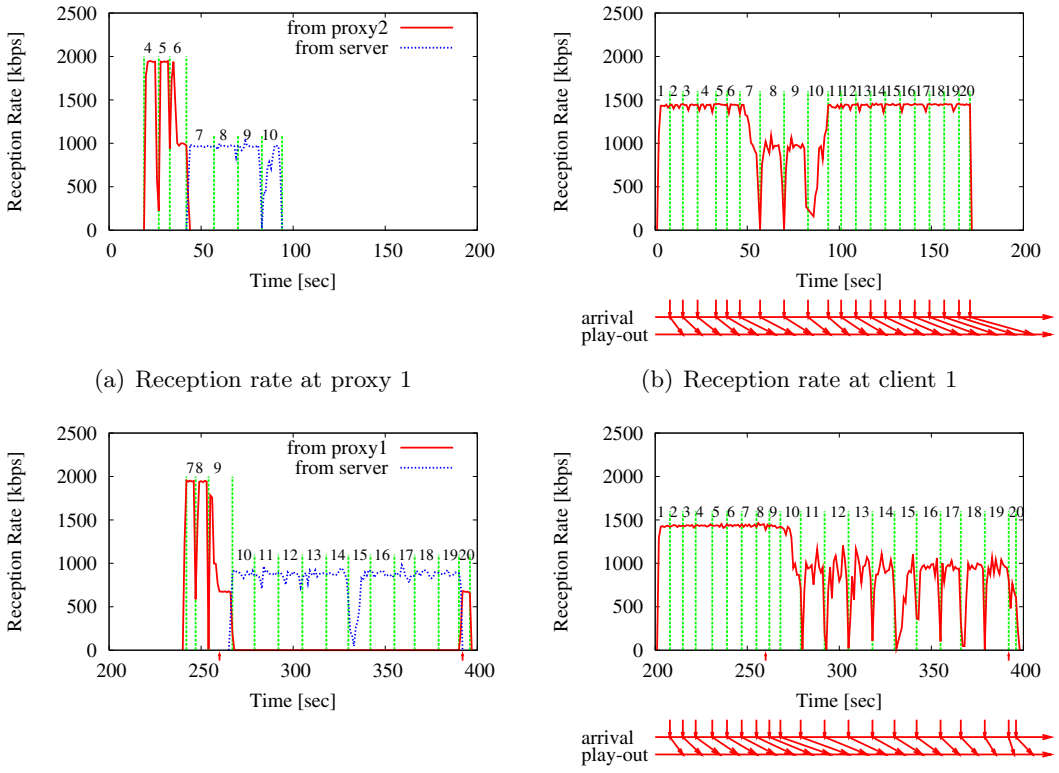
evaluate the block provisioning mechanism when proxies cooperate with each other.

Figure 3.11 illustrates the configuration of our experimental system. For the sake of clarity, we limit the experimental settings to only two proxies and a video server connected through a router. There are two video clients in the system and each is connected to a neighboring proxy through a router. In order to control the delay and link capacity, *NISTNet* is used at the routers. The one-way propagation delay and link capacity are set as shown in Fig. 3.11. In the experiments, we use an MPEG-4 video stream of 200 sec encoded at a rate of 1 Mbps and a frame rate of 30 fps. A block corresponds to 300 VOPs, i.e., 10 sec. Thus, the stream consists of 20 blocks, $b_1, b_2, \cdots, b_{20}$. Initially, proxies already have some blocks as shown in Fig. 3.12. Proxy 1 first has blocks b_1 through b_3 of the coding rate of 1000 kbps, b_4 through b_6 of 700 kbps, and b_{11} through b_{20} of 1000 kbps. Proxy 2 has blocks b_1 through b_6 of 1000 kbps. The window of inquiry I is set to 5.

The experimental scenario is as follows. Client 1 first issues an RTSP OPTIONS message at time 0, and client 2 issues it at 200 sec. Both clients watch the same video stream from the beginning to the end without interactions such as rewinding, pausing, and fast-forwarding. After 260 sec, the link capacity between proxy 1 and proxy 2 is reduced from 2 Mbps to 700 kbps. Using this configuration, we evaluate the capability of the block provisioning mechanism against changes in network conditions and cached blocks on the neighboring proxy. For this purpose, we do not consider other mechanisms such as block prefetching and cache replacement in this experiment. We set the cache buffer capacity to 30 MB, i.e., larger than the size of the whole video stream, and the prefetching window to 0.

3.4.5 Experimental Results

Figures 3.13(a) and 3.13(b) illustrate variations in reception rates observed at proxy 1 and client 1 with *tcpdump*, respectively. First, proxy 1 provides client 1 with cached blocks b_1 to b_3, since they are available fastest and have the highest quality. While sending cached blocks, the proxy can afford to retrieve blocks of higher quality from proxy 2 for blocks b_4 to b_6. Then, proxy 1 retrieves b_7 to b_{10} from the video server. For these 40 sec of blocks, it takes 50 sec of transmission time, because the link capacity between the video server and proxy 2 is smaller than the video rate. Furthermore, the video server sends additional VOPs beginning with the preceding I-VOP, if the specified range starts with a P or B-VOP. This increases the block size and introduces additional delay. However, owing to those cached blocks, proxy 1 has enough time to retrieve them and provide all blocks to client 1 for



(a) Reception rate at proxy 1

(b) Reception rate at client 1

(c) Reception rate at proxy 2

(d) Reception rate at client 2

Figure 3.13: Experimental evaluation of block provisioning mechanism

smooth playback.

In the bottom part of Fig. 3.13(b), instants of block arrivals and those of play-out at client 1 are indicated. In these experiments, the client application first caches a received video block and defers its play-out by 3 sec. As Fig. 3.13(b) illustrates, a user can watch the video without freezes. As a result of block retrieval, the cache table of proxy 1 changes as shown in Fig. 3.12.

Figures 3.13(c) and 3.13(d) illustrate variations in reception rates observed at proxy 2 and client 2, respectively. Proxy 2 first provides client 2 with cached blocks b_1 to b_6. For uncached blocks b_7 to b_{20}, proxy 2 tries retrieving high-quality blocks from proxy 1. At 260 sec, the capacity of the link between proxy 1 and proxy 2 is reduced to 700 kbps. Consequently, proxy 2 contacts the video server from b_9, since the video server can provide the highest quality blocks in the fastest way. However, delays are gradually introduced in

retrieving blocks from the video server due to the insufficient link capacity.

At 392 sec, proxy 2 again contacts proxy 1 to retrieve the block b_{20}. Taking into account the time needed in block transmission from proxy 1 to proxy 2 and that to client 2, the quality of block b_{20} to request to proxy 1 is intentionally reduced to 276 kbps. On receiving the request, proxy 1 applies the video quality adjustment to block b_{20} and transfers the modified block to proxy 2. Proxy 2 caches the block and provides it to client 2. As Fig. 3.13(d) illustrates, all blocks are successfully provided to client 2 through the above mentioned control. Finally, the cache table of proxy 2 becomes as in Fig. 3.12.

In this experiment, not only a single proxy could successfully provide its client with a video stream in time, but also two proxies cooperated to accomplish a continuous video-play out by offering a cached block and the capability of video quality adjustment

3.5 Conclusion

In this chapter, we proposed an effective video streaming mechanism where proxies cooperate with each other. Simulation results showed that our proposed mechanisms can provide users with low-delay and high-quality video streaming services. In addition, we designed and implemented our proxy caching mechanisms on a real system for an MPEG-4 video streaming service employing off-the-shelf and common applications. Through evaluations, it was shown that our proxy caching system can provide users with a continuous video streaming service under dynamically changing network conditions.

Chapter 4

A Traveling Wave-based Communication Mechanism Adaptive to Application Requirements for Wireless Sensor Networks

4.1 Introduction

The development of low-cost microsensor equipments having the capability of wireless communication has caused sensor network technology to attract the attention of many researchers and developers [24-30]. It is possible to obtain information on behavior, condition, and position of elements in a local or remote region by deploying a network of battery-powered sensor nodes there. Each sensor node in such a sensor network has a general purpose processor with a limited computational capability, a small memory, and a radio transceiver.

Due to several restrictions including limited battery capacity, random deployment, and a large number of fragile sensor nodes, a communication mechanism should be energy-efficient, adaptive, robust, fully-distributed, and self-organizing. Furthermore, it should be able to handle various types of communication, i.e. diffusion and gathering, involving the whole

network in accordance with application requirements. For example, a sensor node detecting an emergency would distribute the information over the whole sensor network to alert the other nodes and make them cooperatively react to the emergency. On the contrary, a sensor node detecting an uncertain condition would collect and aggregate sensor information of the other nodes to have a more precise view of the environment by conjecturing from collected information.

There are many proposals of communication mechanism for wireless sensor networks [82-91]. For example, communication mechanisms for data gathering such as LEACH [84, 85], CMLDA [86], and BCDCP [87] cannot function without such global information as the number of sensor nodes deployed, their locations, the predetermined optimal number of clusters, and the residual energy of all sensor nodes. They therefore need additional, possibly expensive and unscalable, communication protocols for collecting and sharing the global information, and they cannot easily adapt to the addition, removal, or movement of sensor nodes.

Furthermore, most of communication mechanisms cannot adopt to dynamically changing application requirements. For example, directed diffusion [89-91] also considers both types of communication, i.e. pull and push. In the two-phase pull diffusion, sinks first emit an *interest* message to find sources. Interest messages are flooded across a network, and matching sources periodically send *exploratory data* to the sink along paths that interest messages traversed. After the initial exploratory data come, the sink chooses one and reinforces the corresponding paths to sources so that following data traverse them to the sink with the smallest latency. The pull-type communication is shown to be appropriate for a case with many sources and few sinks. On the contrary, in the push diffusion, sources first send exploratory data to notify possible sinks of the existence of data. The push-type communication is good for a case with many sinks and few sources. Although directed diffusion can support two different application requirements, these mechanisms cannot be used simultaneously and the mechanism to employ must be determined in advance taking into account expected conditions, including the number of sources and sinks and their communication frequency.

To answer dynamically changing application requirements, a communication mechanism should handle both types of communication, especially in an autonomous and self-organizing manner. In addition, taking into account the insufficient computational capability and memory capacity of inexpensive small sensor nodes, the mechanism must be as simple as

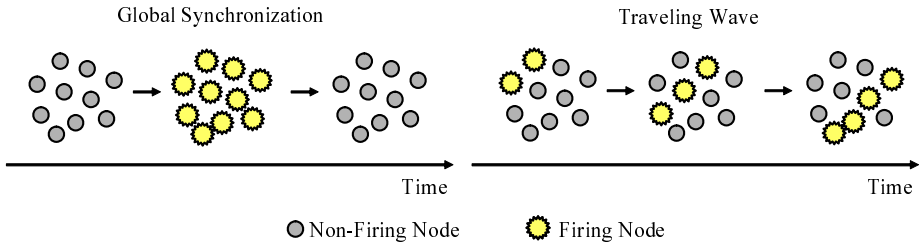


Figure 4.1: Global synchronization and traveling wave

possible. A simple mechanism can also avoid introducing programming and operational errors.

For this purpose, we adopt a pulse-coupled oscillator (PCO) model based on biological mutual synchronization such as that observed in flashing fireflies [49-53]. In a PCO model, synchronous behavior of a group of oscillators is considered. Each oscillator operates on a timer. When the phase of the timer reaches one, an oscillator fires. Oscillators coupled with the firing oscillator are stimulated and they shift the phase of timers by a small amount. Through mutual interactions by stimuli among oscillators, they eventually reach a synchronized behavior. There are several papers which employ a PCO model to make sensor nodes operate in synchrony, e.g., clock synchronization, through a distributed and self-organizing control mechanism [54-65]. In [64, 65], our research group proposed a data gathering mechanism which employ synchronized behavior of a PCO model, and confirmed that it worked in a fully-distributed, self-organizing, robust, adaptive, scalable, and energy-efficient manner.

In this chapter, in contrast to the other works, we focus on another phenomenon observed in a PCO model. In a PCO model, it is shown that not only the global synchronization where all oscillators fire synchronously, but a traveling wave, where oscillators behave synchronously but with fixed phase difference, appears (Fig. 4.1) [52]. By adjusting parameters and functions of a PCO model, we can control the frequency, form, and direction of a wave. We first investigate conditions of a phase response curve (PRC) with which a wireless sensor network reached a preferred phase-lock condition where the phase differences among sensor nodes are kept constant from arbitrary settings of the initial phase of sensor nodes. Next, we propose a self-organizing communication mechanism which generated concentric

traveling waves centered at a sensor node, which wanted to gather information from all sensor nodes or diffuse information to all sensor nodes. In our mechanism, each sensor node broadcasts its sensor information in accordance with the phase of its own timer. When a sensor node receives a radio signal of others, it shifts the phase of its timer. Through mutual interactions among neighboring sensor nodes, they reach the phase-lock and emit sensor information alternately. Through simulation experiments, we confirm that our scheme delivers sensor information to / from designated nodes in a more energy-efficient manner than other method, although it takes time to generate a traveling wave. Furthermore, we implement our mechanism using commercial wireless sensor units, MICAz. Since collisions among synchronized packet emissions affects the performance, we extend the mechanism to distribute timing of packet emission and we confirm that data delivery ratio of about 95 % is accomplished.

The rest of this chapter is organized as follows. First, in Section 4.2, we briefly introduce the mathematical model and its analysis to generate traveling waves. Next, we propose a distributed and self-organizing communication mechanism for wireless sensor networks in Section 4.3, and show simulation results in Section 4.4. Then, we implement, evaluate, and improve our mechanism in Section 4.5. Finally, we conclude this chapter in Section 4.6.

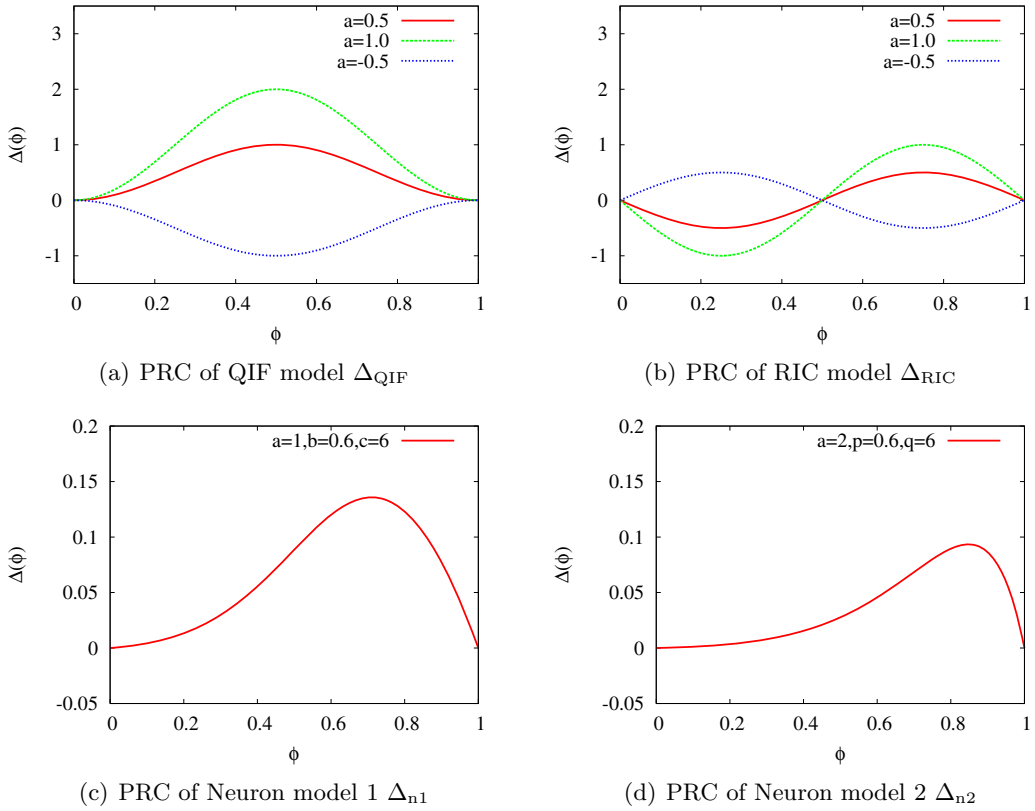
4.2 Analysis of Mathematical Model

In this section, we briefly introduce the PCO model we adopted in this thesis, and investigate conditions of a PRC which leads to a desired form of a traveling wave from arbitrary settings of the initial phase.

4.2.1 Pulse-Coupled Oscillator Model

A PCO model is developed to explain synchronous behaviors of biological oscillators such as pacemaker cells, fireflies, and neurons. In this section, mainly following the model described in [52], we give a brief explanation of the model.

Consider a set of N oscillators. Each oscillator i has phase $\phi_i \in [0, 1]$ ($d\phi_i/dt = 1$). As time passes, ϕ_i shifts toward one and, after reaching it, the oscillator fires and the phase jumps back to zero. Oscillator j coupled with the firing oscillator i is stimulated and



(a) PRC of QIF model Δ_{QIF}

(b) PRC of RIC model Δ_{RIC}

(c) PRC of Neuron model 1 Δ_{n1}

(d) PRC of Neuron model 2 Δ_{n2}

Figure 4.2: PRC examples

advances its phase by an amount $\Delta(\phi_j)$. Thus, we have

$$\phi_j \rightarrow \phi_j + \Delta(\phi_j), \tag{4.1}$$

where $\Delta(\phi)$ is called a phase-response curve (PRC). For example, for the quadratic integrate-and-fire (QIF) model, $\Delta_{\mathrm{QIF}}(\phi) = a(1 - \cos 2\pi\phi)$ (Fig. 4.2(a)) and for the radial isochron clock (RIC) model, $\Delta_{\mathrm{RIC}}(\phi) = -a \sin 2\pi\phi$ (Fig. 4.2(b)) [52]. The PRC of neurons is modeled as $\Delta_{\mathrm{n1}}(\phi) = a\phi(1-\phi)/(1+e^{-c(\phi-b)})$ (Fig. 4.2(c)) or $\Delta_{\mathrm{n2}}(\phi) = a\phi(1-\phi)e^{-p\phi-q(1-\phi)}$ (Fig. 4.2(d)). Here, an oscillator ignores all stimuli at the moment of firing, and an oscillator identifies multiple stimuli received at the same time as one stimulus.

Through mutual interactions, a set of oscillators reach either of the global synchronization where they have the same phase and fire all at once, or the phase-lock condition where

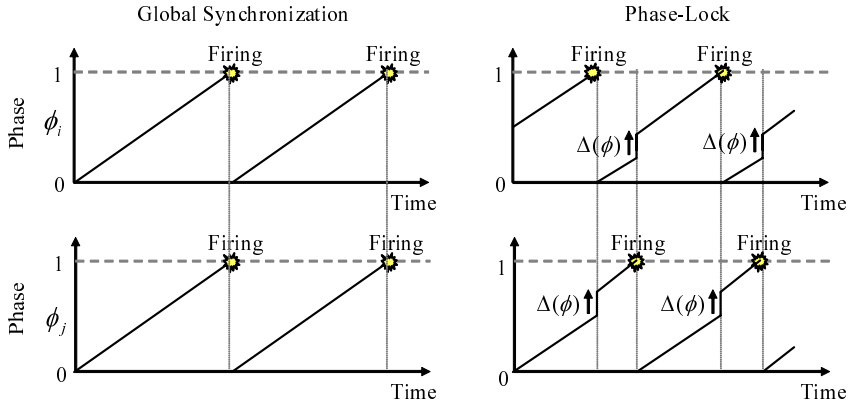
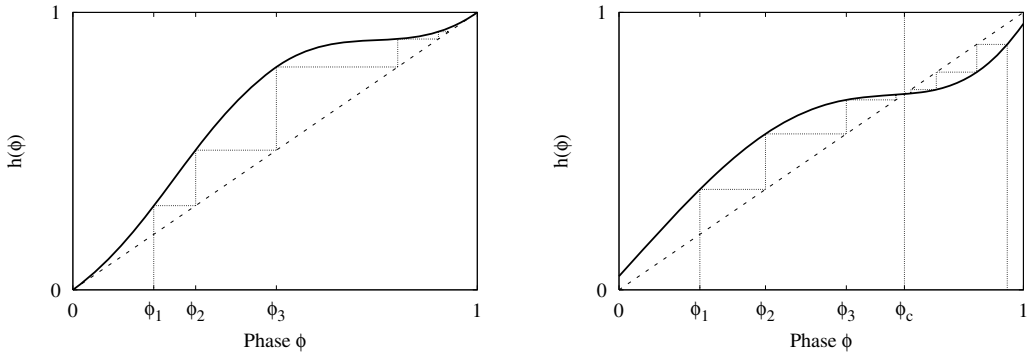


Figure 4.3: Global synchronization and phase-lock



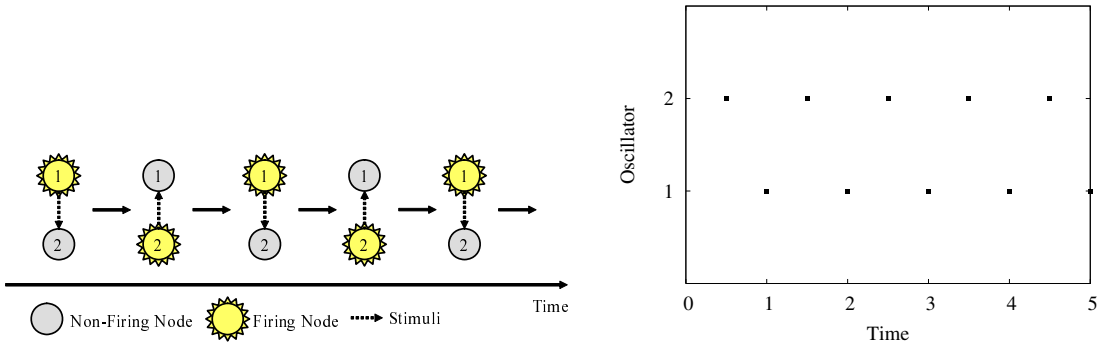
(a) $h(\phi)$ generating the global synchronization

(b) $h(\phi)$ generating the phase-lock

Figure 4.4: Phase transition

phases are different among oscillators with a constant offset as shown in Fig. 4.3. In the case of the phase lock, the geographic propagation of firings seems like a traveling wave as shown in Fig. 4.1.

Whether a network reaches the global synchronization or the phase-lock depends on the initial phase of timers or properties of the PRC [92]. In Fig. 4.4, $h(\phi)$ indicates the phase at which an oscillator is stimulated again by a neighboring oscillator, after the oscillator is stimulated from a neighboring oscillator at the phase of ϕ. For example, in the case of a pair of oscillators, it is defined as $h(\phi) = 1 - F(1 - F(\phi))$ where $F(\phi) = \phi + \Delta(\phi)$. A dotted diagonal line stands for $h(\phi) = \phi$, and a stepwise line stands for phase transition of



(a) Firing pattern in two oscillators (b) Timing of firing in two oscillators

Figure 4.5: Traveling wave in two oscillators

an oscillator whose initial phase is ϕ_1. When an oscillator is stimulated at ϕ_1, the oscillator changes its phase by using Eq. (4.1), and it will observe the next fire and be stimulated at ϕ_2.

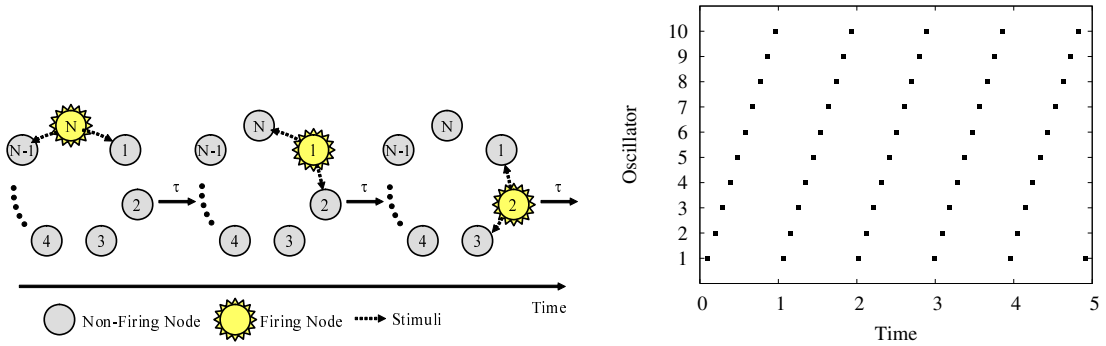
In Fig. 4.4(a), through being stimulated several times, the phase $h(\phi)$ becomes one from arbitrary initial phase. $h(\phi) = 1$ means that an oscillator receives a stimulus from another firing oscillator when the oscillator itself is firing. Therefore, they fire at the same time. On the contrary, if oscillators have a PRC corresponding to Fig. 4.4(b), $h(\phi)$ converges at ϕ_c independently of the initial phase. It means that all oscillators reach the condition where the phase is always ϕ_c when being stimulated. Therefore, oscillators fire with the time difference of ϕ_c at the stable condition.

4.2.2 Generation of Various Traveling Waves

In this section, we investigate initial conditions that lead to desired phase-lock conditions, i.e. traveling waves. We start from the simplest case, two alternately firing oscillators, then move to a ring, a line, two types of concentric circles, a wedge, and a radar-shaped traveling wave.

Traveling Wave in Two Oscillators

First, we consider phase-lock condition in a pair of oscillators as shown in Fig. 4.5(a) [52]. Suppose that when oscillator 1 fires at time 0, oscillator 2 is at ϕ_2 so that the new phase for oscillator 2 becomes $F(\phi_2)$. At $t_1 = 1 - F(\phi_2)$ oscillator 2 fires, then oscillator 1 at



(a) Firing pattern in a ring

(b) Timing of firing in a ring ($N = 10$)

Figure 4.6: Ring-type traveling wave

$\phi_1 = t_1 = 1 - F(\phi_2)$ moves to the new phase $F(t_1) = F(1 - F(\phi_2))$. At $t_2 = 1 - F(t_1)$ oscillator 1 fires once again, and the phase of oscillator 2 is $1-F(t_1) = 1-F(1-F(\phi_2))$. To have the phase-lock condition, $\phi_2 = 1-F(1-F(\phi_2))$. Consequently, when initial conditions are comply with $\phi_f = 1 - F(1 - F(\phi_f))$, oscillators fire alternately. In the case of $\phi^{'} = 1$, the occurrence condition is $|\phi_1 - \phi_2| = 1 - \phi_f$.

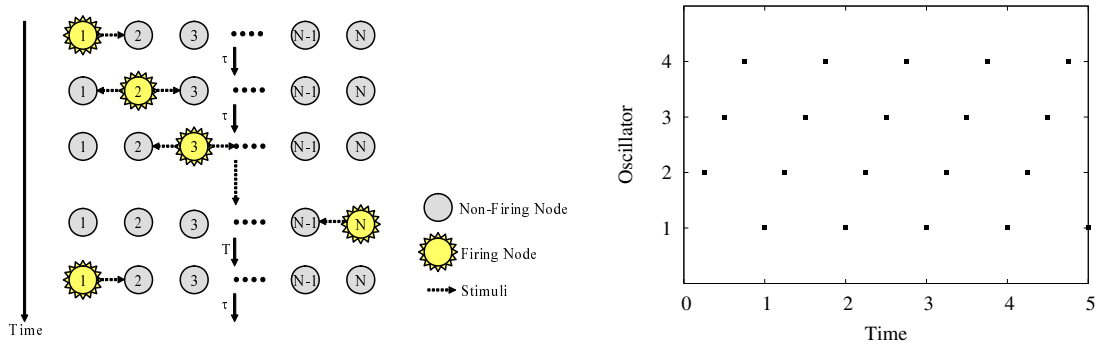
Figure 4.5(b) shows the timing of firing in two oscillators. We used the RIC PRC with $a = 0.1$. ϕ_1 and ϕ_2 were set at 0 and 0.5, respectively. In Fig. 4.5(b), we can see that oscillators fire alternately. X-axis corresponds to time and y-axis corresponds to identifiers of oscillators. Each dot stands for the timing that an oscillator fires.

Ring-Type Traveling Wave

Next, we consider the case of a ring of N oscillators as shown in Fig. 4.6(a) [52]. Since an oscillator is stimulated by two neighboring oscillators, Eq. (4.1) becomes as

$$\phi_i^{'} = 1 + \Delta(\phi_i)[\delta(\phi_{i-1}) + \delta(\phi_{i+1}) - \delta(\phi_{i-1})\delta(\phi_{i+1})], \tag{4.2}$$

where we identify 0 with N and $N+1$ with 1. Consider oscillators fire in order of $1 \rightarrow 2 \rightarrow \cdots \rightarrow N$ at constant phase-difference τ. When oscillator N fires, oscillator $N-1$ and 1 are stimulated. At this time, the phase of oscillator $N-1$ is τ and its new phase becomes $F(\tau)$. Oscillator 1 is at $(N-2)\tau + F(\tau)$ and its new phase ϕ_1 becomes $F((N-2)\tau + F(\tau))$. Here, $\phi_1 = 1 - \tau$ holds because oscillator 1 will fire after τ. Therefore, the phase of each



(a) Firing pattern in a line

(b) Timing of firing in a line ($N = 4$)

Figure 4.7: Line-type traveling wave

oscillators in a traveling ring wave should satisfy the following conditions.

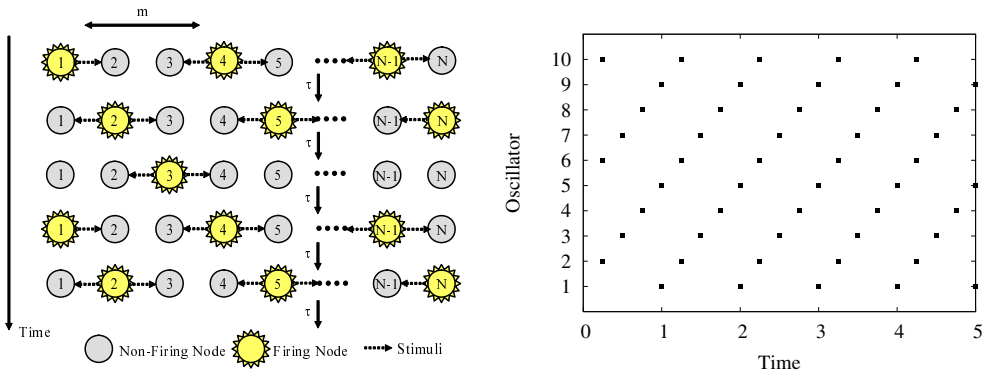
$$\begin{aligned}
\phi_{N-1} &= F(\tau) \\
\phi_{N-2} &= \tau + F(\tau) \\
\vdots &= \vdots \\
\phi_i &= (N-i-1)\tau + F(\tau) \\
\vdots &= \vdots \\
\phi_2 &= (N-3)\tau + F(\tau) \\
\phi_1 &= F((N-2)\tau + F(\tau)) = 1 - \tau.
\end{aligned}$$

From this, we have the following formula.

$$F((N-2)\tau + F(\tau)) + \tau = 1. \tag{4.3}$$

Waves with multiple cycle replace the 1 with m.

Figure 4.6(b) shows the timing of firing in a ring of oscillators, where $N = 10$. We used the RIC PRC with $a = 0.1$. τ was set at 0.0964 derived from Eq. (4.3). In Fig. 4.6(b), we can see that a fire travels along a ring.



(a) Multiple firing pattern in a line (b) Timing of multiple firing in a line ($N = 10$)

Figure 4.8: Line-type multiple traveling wave

Line-Type Traveling Wave

In this section, we consider a line of N oscillators. Consider oscillators fire in order of $1 \to 2 \to \cdots \to N$ at constant phase-difference τ, and oscillator 1 fires after T unit of time from a fire of oscillator N as illustrated in Fig. 4.7(a) where dashed arrows stand for stimuli given by a firing oscillator to neighboring oscillators.

When oscillator i $(1 \leq i < N-1)$ fires, the new phase of oscillator $i+1$ becomes

$$F((N-3)\tau + T + F(\tau)) = 1 - \tau. \tag{4.4}$$

Similarly, when oscillator $N-1$ fires, the new phase of oscillator N becomes

$$F((N-2)\tau + T) = 1 - \tau. \tag{4.5}$$

Finally, when oscillator N fires, the new phase of oscillator 1 becomes

$$(N-2)\tau + F(\tau) = 1 - T. \tag{4.6}$$

Equations (4.4) through (4.6) describe the condition for the existence of traveling waves on a line of oscillators. Figure 4.7(b) shows the timing of firing in a line, where $N = 4$, $T = 0.25$, and $\tau = 0.25$. We used $\Delta(\phi) = -a \sin 4\pi\phi$ as PRC, where $a = 0.05$.

Next, we consider another pattern of traveling wave in a line illustrated in Fig. 4.8(a).

In this case, oscillators which are distant by m (m is a natural number) fire at the same time. Suppose that oscillator i ($1 < i < N$) fires at time 0. At time τ, oscillator $i+1$ fires and the new phase of oscillator i becomes $F(\tau)$. After $(m-1)\tau$ unit of time, oscillator $i-1$ fires, and the new phase of oscillator i becomes,

$$F((m-1)\tau + F(\tau)) \quad = \quad 1 - \tau. \tag{4.7}$$

Similarly, let us consider oscillator 1. Oscillator 1 fires at time 0. At time τ, oscillator 2 fires and the new phase of oscillator 1 becomes $F(\tau)$. After $m\tau$, oscillator 1 will fire again. Therefore,

$$(m-1)\tau + F(\tau) \quad = \quad 1 - \tau. \tag{4.8}$$

Finally, we consider oscillator N. Oscillator $N-1$ fires after $m\tau$ from oscillator N fires.

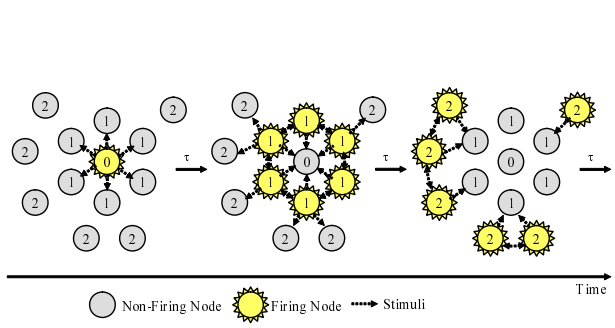
$$F(m\tau) \quad = \quad 1 - \tau. \tag{4.9}$$

Equations (4.7) through (4.9) describe the condition for the existence of traveling waves. We should note that Eqs. (4.7) through (4.9) are identical to Eqs. (4.4) through (4.6) when m is equal to $N-1$ and T is equal to τ.

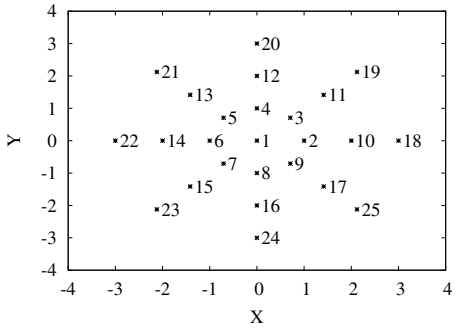
Figure 4.8(b) shows the timing of firing in a line, where $N = 10$, $m = 3$, and $\tau = 0.25$. In Fig. 4.8(b), two to three oscillators fire at the same time as $(2, 6, 10) \rightarrow (3, 7) \rightarrow (4, 8) \rightarrow (1, 5, 9) \rightarrow (2, 6, 10)$.

Concentric Circle-Type Traveling Wave

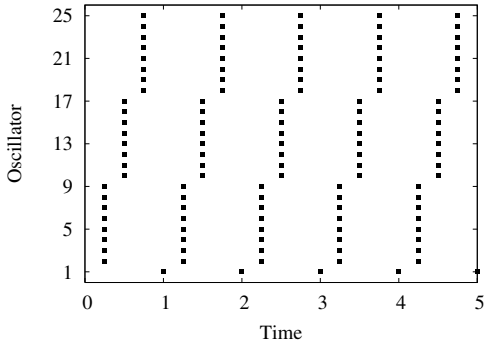
In this section, we consider a traveling wave drawing a concentric circle as in [64]. Figure 4.9(a) illustrates interactions among firing oscillators in concentric circles. The number in each circle, i.e. an oscillator, indicates the number of hops from the center of circles called level in [64]. Oscillators fires in order of levels. We assume an oscillator ignores all stimuli at the moment of firing [52], and an oscillator identifies multiple stimuli received at the same time as one stimulus. Following this assumption, we can regard oscillators on the same level as one oscillator. Therefore, we can apply the same condition derived in Section 4.2.2 by defining the same initial condition for oscillators on the same level.



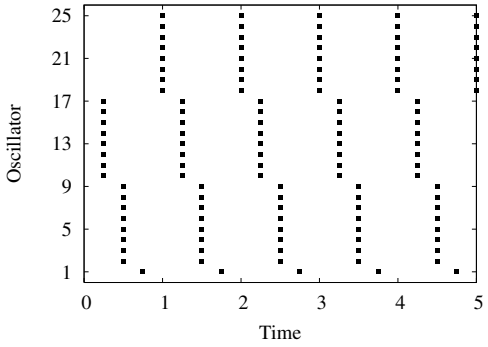
(a) Firing pattern in a concentric circle



(b) Location of oscillators



(c) Timing of firing

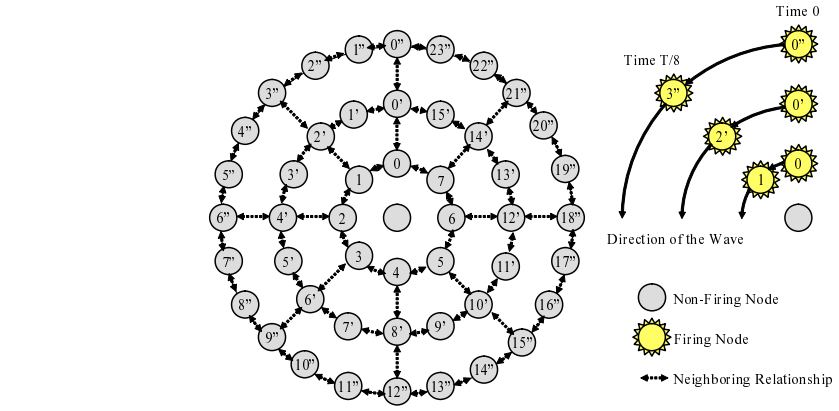


(d) Timing of firing in reverse

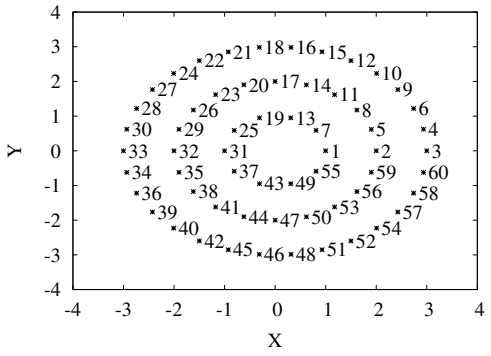
Figure 4.9: Concentric circle-type traveling wave

We confirmed the existence of phase-lock condition to generate a concentric circle-shaped traveling wave. Figure 4.9(b) illustrates the simulated network of 25 oscillators. For easier understanding, oscillators are placed to form concentric circles. However, we can generate a traveling wave of the form of a concentric circle on a sensor network with arbitrary node distribution. Oscillators are numbered from the center to the edge. The oscillator at the center is named 1. Oscillators from 2 to 9 are on the most inner circle which has a radius of one unit of distance. Oscillators from 10 to 17 are on the middle circle, and ones from 18 to 25 are on the third. Each oscillator interacts with all other oscillators that are within distance of 1.5. We used $T = 0.25$ and $\tau = 0.25$ on Eqs. (4.4) through (4.6). In Fig. 4.9(c), we can observe a traveling wave propagating from the center toward the edge where all oscillators on the same circumference fire synchronously.

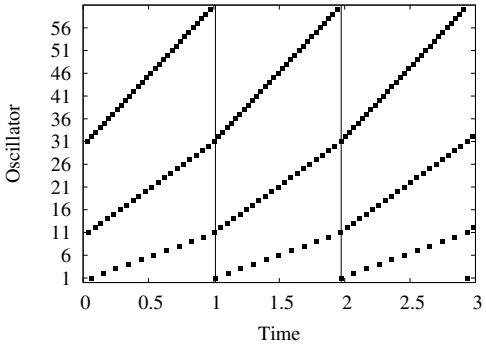
When we give the initial conditions of oscillators in reverse, a wave propagates from the edge toward the center as shown in Fig. 4.9(d).



(a) Firing pattern in a radar



(b) Location of oscillators



(c) Timing of firing

Figure 4.10: Radar-type traveling wave

Radar-Type Traveling Wave

In this section, we consider a radar-shaped traveling wave as illustrated in Fig. 4.10(a). The number in each circle indicates the order of firing on a circumference, and we call it as level. Oscillators with the same level value on different circumferences do not necessarily fire simultaneously. On the contrary, oscillators with different level values on different circumferences fire simultaneously if they are on the same radius. For example, at time 0, oscillators 0, $0'$, and $0''$ on the same radius fire simultaneously. When we consider a cycle of T, $T/8$ unit of time later, oscillator 1, $2'$, and $3''$ fire at the same time. Between them, oscillator $1'$ fires at $T/16$, and oscillators $1''$ and $2''$ fires at $T/24$ and $T/12$, respectively.

A radar-shaped traveling wave can be generated by first organizing oscillators into concentric circles. Next, on each of a circumference, a ring-shaped traveling wave is generated

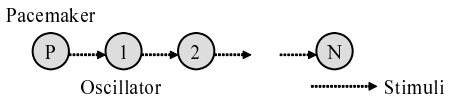


Figure 4.11: Oscillators in tandem

while making oscillators on a radius fire simultaneously. For this purpose, we assume that an oscillator receives stimuli only from neighboring oscillators on the same circumference and those in the same radius as shown by dashed arrows in the figure. In addition, we assume that the center node does not fire, or oscillators on the most inner circle ignore firing of the center node.

Figure 4.10(b) illustrates the simulated network of 60 oscillators. Oscillators from 1 to 10 are on the most inner circle which has a radius of one unit of distance. Oscillators from 11 to 30 are on the middle circle, and ones from 31 to 60 are on the third. Each oscillator interacts with all other oscillators that are within distance of 1. Derived from Eq. (4.3), τ was set at 0.0964, 0.0482, and 0.0323 for the most inner, the middle, and the third circle, respectively. Figure 4.10(c) shows the timing of firing. A solid line indicates the time that oscillator 1 fired. In Fig. 4.10(c), we can observe a radar-shaped traveling wave where oscillators on the same circumference fire in order the same time and those on the same radius fire at.

4.2.3 Condition of PRC to Generate Traveling Waves

In previous section, we showed that we could generate various traveling waves by using PCO model. However, it was sensitive to the initial phase setting. In this section, we investigate conditions of PRC that lead to desired phase-lock condition regardless of the initial phase to generate preferred traveling waves. We call an oscillator which dominates and controls a PCO network as a pacemaker. To keep the timing and frequency of communication, a pacemaker will not be stimulated and will fire at regular intervals, which corresponds to the data gathering or diffusion cycle in a wireless sensor network.

Oscillators in Tandem

First, we consider a traveling wave in a PCO network where oscillators are arranged in a line as shown in Fig. 4.11. Each circle stands for an oscillator, each arrow shows the direction

of stimuli, and oscillators are numbered by the number of hops from the pacemaker. An oscillator is stimulated only by its neighboring oscillator which is closer to the pacemaker. A pacemaker fires periodically at regular intervals of one time unit. Oscillators fire in order of the pacemaker, oscillator 1, oscillator 2, $\cdots$, oscillator N at constant phase-difference τ. Therefore, if a pacemaker fires at time 0, oscillator 1 fires at time τ, and oscillator N fires at time $N\tau$. Here, we consider $0 < \tau < 1$.

Now, consider phase transitions of oscillators at the phase-lock condition. Assume that after t time unit since oscillator i ($1 \leq i \leq N$) fired, an oscillator i is stimulated by oscillator $i-1$. Oscillator 0 corresponds to the pacemaker. Since oscillators fire at constant phase-difference τ, the phase of an oscillator becomes $1-\tau$ when it is stimulated by a neighboring oscillator, i.e. $F(t) = 1-\tau$. Then, oscillator i fires at $\tau + t$. Since an oscillator fires at regular intervals of one at the phase-lock condition, we have $t + \tau = 1$. Hence, we have

$$\Delta(1-\tau) = 0. \tag{4.10}$$

To generate a desired traveling wave regardless of the initial phase, an oscillator should advance its phase towards $1-\tau$ when it is stimulated during $0 \leq \phi < 1-\tau$, and push back its phase towards $1-\tau$ when it is stimulated during $1-\tau < \phi < 1$. Finally, we have following conditions of PRC to generate a traveling wave regardless of the initial phase.

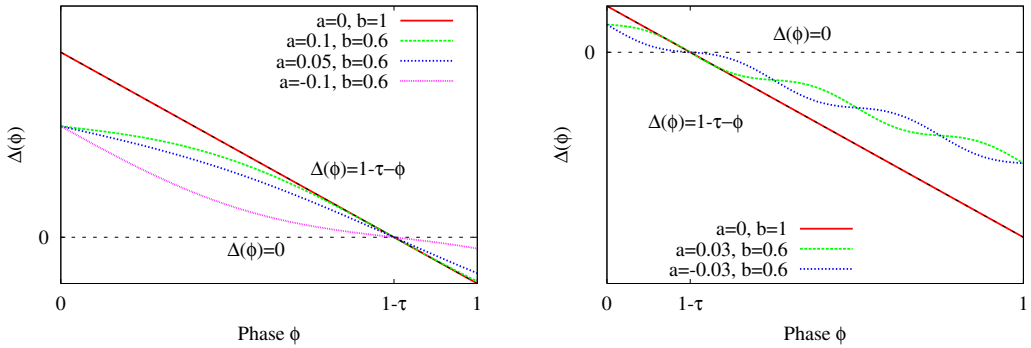
$$\begin{cases} 0 < \Delta(\phi) \leq 1-\tau-\phi & (0 \leq \phi < 1-\tau) \\ \Delta(\phi) = 0 & (\phi = 1-\tau) \\ 1-\tau-\phi \leq \Delta(\phi) < 0 & (1-\tau < \phi < 1). \end{cases} \tag{4.11}$$

For example, following PRC function satisfies Eq. (4.11).

$$\Delta_s(\phi) \quad = \quad a \sin \frac{\pi}{1-\tau}\phi + b(1-\tau-\phi) \tag{4.12}$$

Here, a ($-b(1-\tau)/\pi < a \leq (1-b)(1-\tau)/\pi$) and b ($0 < b \leq 1$) are parameters which determine characteristics of PRC. Figure 4.12 illustrates PRC $\Delta_s(\phi)$ for two different settings of a and b when $\tau = 0.2$. Two dot-and-dash lines stand for $\Delta(\phi) = 0$ and $\Delta(\phi) = 1-\tau-\phi$, respectively. The curve of PRC satisfying Eq. (4.11) must lie between these two lines.

With a PRC satisfying the above conditions, oscillators fire in order of the pacemaker,



(a) In the case of $0 < \tau < 0.5$ (b) In the case of $0.5 < \tau < 1$

Figure 4.12: PRC Δ_s from Eq. (4.12)

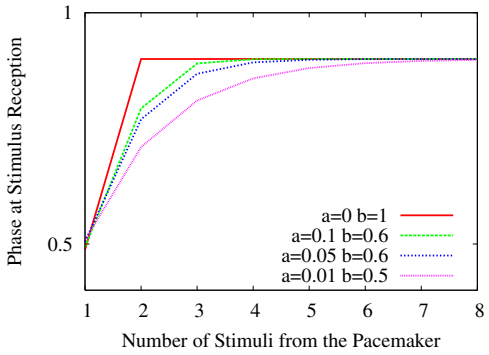


Figure 4.13: Phase transition of oscillator 1

oscillator 1, oscillator 2, $\cdots$, oscillator N at constant phase-difference of τ at the phase-lock condition. This can also be regarded as a traveling wave propagating from oscillator N toward the pacemaker, with constant phase-difference $1 - \tau$. Therefore, to have a diffusion type of communication, where information propagates from the peacemaker to oscillator N with constant phase-difference τ, τ should be set as $\tau < 0.5$. On the contrary, to have a gathering type of communication, τ should be set as $\tau > 0.5$.

Figure 4.13 shows a phase of oscillator 1 when it receives a stimulus from the pacemaker where $\tau = 0.1$ and $N = 1$. The initial phase of oscillator 1 is randomly chosen, and results are averaged over 1000 simulations. At $a \neq 0$, as parameters a and b increase, a traveling wave emerges more rapidly. Especially, a traveling wave emerges by only one

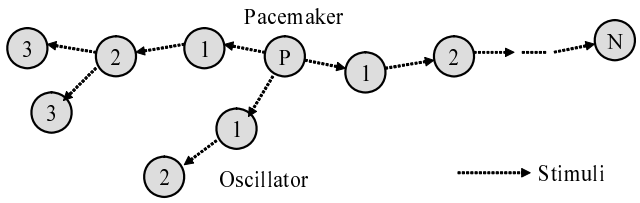


Figure 4.14: Two-dimensional arrangement of oscillators

interaction, i.e. stimulus, among oscillators with $a = 0$ and $b = 1$. However, such aggressive setting spoils the resilience of the mechanism against a failure of node and unexpected influence from the environment, since a single firing emitted at a wrong time will drastically change the state of the whole system. Therefore, a PRC function and its parameters should be appropriately determined taking into account the trade-off between the speed that a traveling wave emerges and the resilience against failures.

Oscillators in Two-Dimensional Arrangement

A PRC satisfying Eq. (4.11) can also be applied to the case of two dimensional arrangement of oscillators. By making a tree whose root is the pacemaker and setting the direction of stimuli as shown in Fig. 4.14, we can adopt the same PRC and generate a traveling wave propagating from or to the pacemaker in a two-dimensional area. Although any routing protocol for wireless sensor networks is viable to organize such tree-type topology, a simple way of setting such relationship among oscillators will be given in the next section.

4.3 A Traveling Wave-based Communication Mechanism

In this section, we propose a fully-distributed and self-organizing communication mechanism for wireless sensor networks. In our mechanism, any of sensor nodes can become a point, called core node, from which messages are disseminated or to which messages are gathered in accordance with application requirements. Core node plays a role of a pacemaker in the PCO model.

4.3.1 Basic Behavior

Sensor node i ($1 \leq i \leq N$) has a timer with phase $\phi_i \in [0, 1]$. It maintains PRC function $\Delta(\phi)$, level value l_i, session identifier s_i, direction δ_i, and offset τ ($0 < \tau < 0.5$). Initially a level value, a session identifier, and a direction are set to zero. A level value indicates the number of hops from the core node and it is used to define the relationship among sensor nodes. Direction δ_i is a parameter which controls the direction of information propagation, and it is set at 1 for diffusion and -1 for gathering. The offset defines the interval of message emission between a node of level $l-1$ and that of level l. The PRC function and offset are determined at the deployment phase, but the offset can be dynamically adjusted as explained later. In this thesis, based on Eq. (4.12), we use the following PRC function for all sensor nodes.

$$\Delta(\phi) \quad = \quad a \sin \frac{\pi}{g}\phi + b(g - \phi), \tag{4.13}$$

Here, g is defined as $(1 - \delta_i\tau) \bmod 1$.

As time passes, phase ϕ_i shifts toward one and, after reaching it, sensor node i broadcasts a message and the phase jumps back to zero. A message that sensor node i emits contains level value l_i, session identifier s_i, direction δ_i, and its information aggregated with other sensor's information kept in its buffer. To initiate a new communication, a core node broadcasts a message containing a new session identifier set at the current value plus one, a level value of zero, the direction, and information to disseminate or gather.

Now, sensor node i receives a message from sensor node j. If session identifier s_j is larger than s_i, sensor node i considers that a new communication begins. Therefore, it sets its level value l_i at $l_j + 1$, session identifier s_i at s_j, and direction δ_i at δ_j. Then, it is stimulated to join a new traveling wave. This mechanism means that the current communication is terminated by a newly initiated communication. To avoid unintended termination of communication by other sensor nodes, a core node might advertise its desired communication period in a message it emits. However, it requires an additional mechanism such as clock synchronization, and it is left as one of future research issues. If session identifiers are the same but the level value l_j is smaller than l_i, sensor node i sets its level value l_i at $l_j + 1$, direction δ_i at δ_j, and it is stimulated. Stimulated sensor node i shifts its phase based on the PRC function. As in the PCO model, a sensor node is not stimulated by messages from sensor nodes with a smaller level value during the following duration of τ

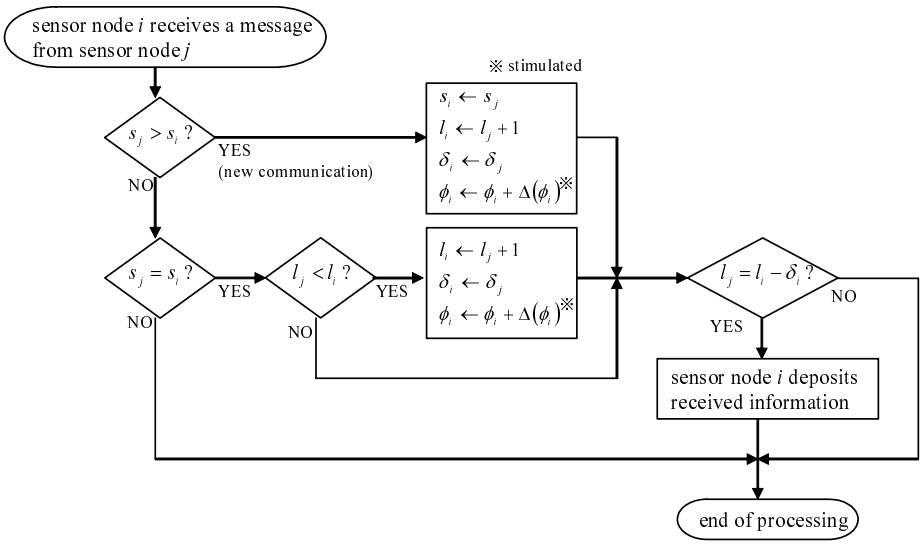


Figure 4.15: Node behavior on message reception

when it has already been stimulated, to avoid being stimulated by deferred messages. If the session identifier is the same and level value l_j is $l_i - \delta_i$, sensor node j is an upstream node of sensor node i. Therefore, to relay information of sensor node j to the next downstream node, sensor node i deposits the received information in its local buffer. If a message does not satisfy any of the above conditions, sensor node i ignores it. We should note here that a sensor node only emits a message in accordance with the phase of its timer. No additional message is required to organize a traveling wave. The algorithm is illustrated in Fig. 4.15.

4.3.2 Power-Saving Mode

Through mutual interactions among neighboring sensor nodes, they reach the phase-lock, and a sensor node moves to a power-saving mode. In power-saving mode, a sensor node wakes up when its phase is at $1 - \tau$ to receive messages from upstream nodes. Upstream nodes are scheduled to emit their messages from $1 - \tau$ to 1. When its phase reaches one, a sensor node broadcasts a message. After that, it keeps awake for τ to receive messages from downstream nodes, and then goes to sleep by turning off its radio transceiver and other needless modules.

Here, τ should be appropriately determined considering trade-off between the rate of successful message reception and the lifetime of sensor network. The smaller τ is, the smaller

probability of successful message reception by missing messages delayed by collisions in radio signals. At the same time, a smaller τ leads to longer lifetime of sensor network, since a node is awake for the duration of 2τ in one communication cycle.

To judge whether the phase-lock condition is globally accomplished or not, we consider T_{max} as the worst-case time required for a sensor node to establish the phase-lock condition with a neighboring node closer to the core node. We can expect that a sensor node can move to a power-saving mode after $(l_i + (1 - \delta_i)/2) \times T_{max}$ since the level value is updated.

If the phase-lock condition is lost for some reasons after a power-saving mode is activated, a sensor node does not receive any valid message when it is awake. In such a case, a sensor node stops a power-saving mode to reorganize the phase-lock condition.

4.3.3 Addition and Removal of Sensor Nodes

Next, we consider the case where a new sensor node is introduced in a sensor network in operation. Initially, the session identifier of a new sensor node is set at zero. Therefore, it does node affect other sensor nodes. Being stimulated several times, its level value, session identifier, and direction are correctly identified, and its timer synchronizes at constant phase-difference with that of a neighboring sensor node whose level is smaller by one.

On the contrary, when a sensor node disappears due to battery depletion or removal, a sensor node that is synchronized with the vanished node will be stimulated by another of the same level as the vanishing node. If there is no other node with a smaller level value in its vicinity, the sensor node becomes isolated. Since it does not receive stimuli any more, it can recognize the isolation and then it initializes its session identifier so that it can synchronize with other neighboring sensor nodes.

4.3.4 Multiple Core Nodes

In addition, we consider the case that there are two or more core nodes with the same session identifier in a sensor network. Since it takes time for information to propagate between the edge of a wireless sensor network and a core node, it is a good idea to have multiple core nodes for one communication to solve the scalability problem. In such case, the sensor network is divided into clusters each of which has one core node. Each core node can gather or diffuse information in its cluster.

A sensor node which is at the same hop count from more than one core node, which

we call a border node, receives messages from different clusters. Since session initiation is not necessarily synchronized among core nodes and time required for stimulus propagation would differ among paths from core nodes, such multiple stimuli prevent a border node from establishing the phase-lock condition with neighboring nodes.

Therefore, a border node chooses one cluster which it belongs to. First, after a level value is updated, a border node waits for the duration of $(l_i - 1) \times T_{max} + \tau$ until the phase-lock condition is established in each of clusters. Then, it begins to stick on the timing of the first message it receives. To avoid being stimulated by deferred messages or message originated from another core node, it ignores received stimuli during the following duration of $1 - \tau$ when it has already been stimulated.

4.3.5 Node Failures

Finally, we consider cases of node failures. First, the failure of a radio transmitter has no influence on other communication, since a failed sensor node can not emit any message and never stimulate neighboring nodes. On the other hand, a sensor node with a failed receiver keeps sending messages based on its timer. The timer of a failed node keeps its own pace independently of the others. Therefore, when the phase-lock condition is not established yet, the failed node disturbs establishment of the phase-lock condition by stimulating neighboring nodes at inappropriate timings. However, a failed node eventually considers it is isolated for not receiving any message from neighboring nodes. Then, it initializes its state and it does not affect others anymore.

In some cases, a timer gains or loses, being affected by, for example, geomagnetism. Basically, a wrong timer will be correctly adjusted from stimulations. A sensor node with a timer which gains, stimulates neighboring nodes at a wrong timing, since sensor nodes take the first message it receives and ignores the following delayed messages. If a wrong timer keeps an advanced phase, the problem is that the interval between message emission of the failed node and its upstream node becomes smaller than τ and it does not bring any serious influence on message propagation.

On the other hand, a sensor node with a timer which loses does not affect the phase-lock condition very much. If a sensor node which is stimulated by a failed node has another normal node with a smaller level value, it is always stimulated by the normal sensor node and ignores delayed messages from the failed node. Otherwise, the interval of message emission of the failed sensor node and the affected sensor node becomes longer than τ.

In addition, we consider wrong setting of parameters such as δ_i, l_i, and s_i of sensor node i by temporal error of memory or CPU. Direction δ_i is updated periodically when a sensor node receives a message from a sensor node with a smaller level value.

When level value l_i is incorrectly larger than the actual hop count from a core node, messages emitted by sensor node i do not affect neighboring sensor nodes with a smaller level value. On the contrary, if the level value is too small, neighboring nodes would wrongly identify their distance from the core node. It first disturbs establishing the phase-lock condition. However, since the level value of the failed sensor node is the smallest in its range of radio signals, it does not receive any stimulus, i.e. a message with a further larger level value. Therefore, it considers it is isolated and initializes its session identifier so that its level value will be adjusted correctly.

When the session identifier s_i is incorrectly smaller than the current session identifier used in a wireless sensor network, the failed sensor node does not affect the others at all. Its session identifier will be corrected on receiving a message from a neighboring sensor node. On the contrary, a larger session identifier s_i means that a new communication is initiated by the failed sensor node. Since the other nodes cannot judge whether a new communication is actually initiated or not, it is handled as normal. When another sensor node initiates a new communication, a new session identifier is used and the failed node does not affect the others any more.

4.4 Simulation Experiments

In this section, we show results of simulation experiments. In the simulation, the range of the radio signal is fixed at 2 units of length. The initial phase of sensor node is randomly chosen. A sensor node consumes 81 mW for transmission, 30 mW for receiving and idle, and 0.003 mW for sleep [66]. Initial energy is 50 J for all nodes. We use Eq. (4.13) with $a = 0.01$ and $b = 0.5$ as the PRC function and τ is set at 0.1.

4.4.1 Basic Behavior

We first confirm the basic behavior of our communication mechanism. We consider sensor networks of 100 sensor nodes randomly distributed in a 10×10 region as shown in Fig. 4.16. From 0 to 20 time units, we randomly chose a sensor node A as a core node for information diffusion. Then, from 20 to 40 time units, we randomly chose another sensor node B as a

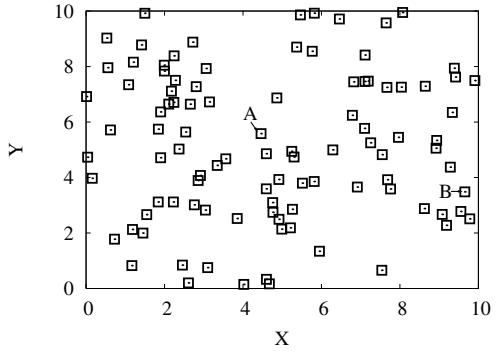
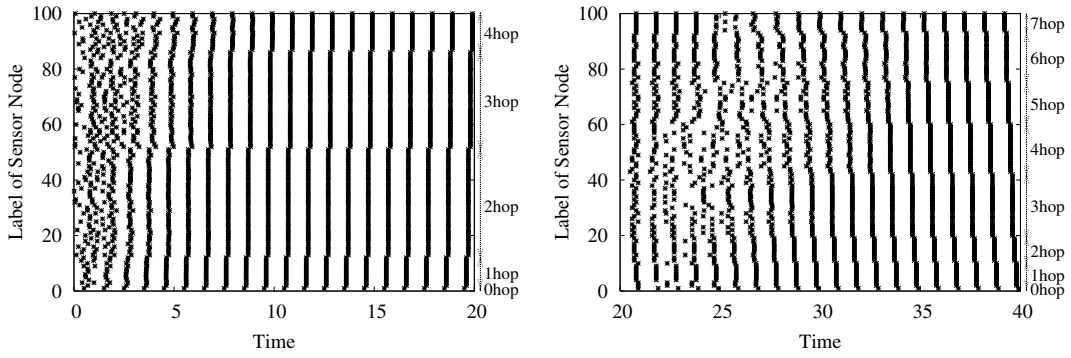


Figure 4.16: Sensor distribution in the simulation experiments



(a) Diffusion (b) Gathering

Figure 4.17: Timing of message emissions

core node for information gathering.

Figure 4.17 shows how the sensor network reached the phase-lock condition in a certain simulation experiment. Each mark stands for an instant when a sensor node emitted a message. For easier understanding, sensor nodes are sorted in order of the hop count from the core node. In Fig. 4.17(a), at first, all sensor nodes randomly and independently emit messages. However, by exchanging stimuli several times, the phase-lock condition is eventually accomplished and a regular pattern appears. It is clearly shown that message emission is in order of the hop count of sensor nodes, from the core node to nodes with larger numbers. In Fig 4.17(b), it is shown that the phase-lock condition for information diffusion is first broken for information gathering initiated by sensor node B. Then, although it takes

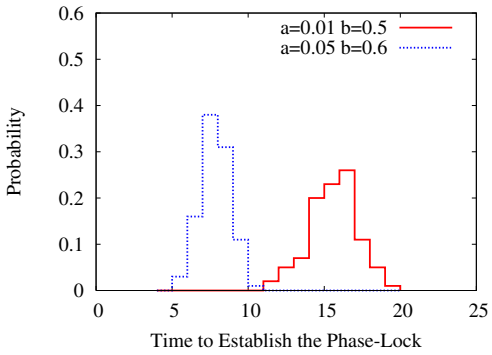


Figure 4.18: Distribution of the time to establish the phase-lock condition

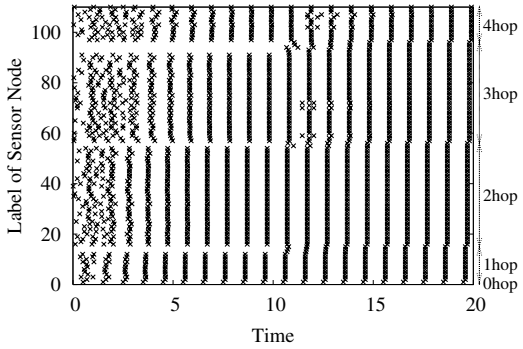


Figure 4.19: Timing of message emissions with dynamic deployment

longer time than for diffusion, the new phase-lock condition appears, where information propagates from the edge of the sensor network towards the core node B.

Over 100 experiments, the average time to establish the phase-lock condition is 15.5 time units. The time ranges from 11.6 to 19.6 depending on the distribution of sensor nodes, location of the core node, and phase of sensor nodes. The histogram is shown in Fig. 4.18. The time to reach the stable phase-lock condition can be reduced by using a set of larger a and b satisfying Eq. (4.11). For example, with $a = 0.05$ and $b = 0.6$, the minimum, average, and maximum are 5.96, 8.10, and 10.7, respectively.

We confirmed that traveling waves can be formed in a wireless network with addition, movement, and removal of sensor nodes, and failed nodes. In Fig. 4.19, an example of the case of node addition is illustrated. At 10 time units, after the phase-lock condition

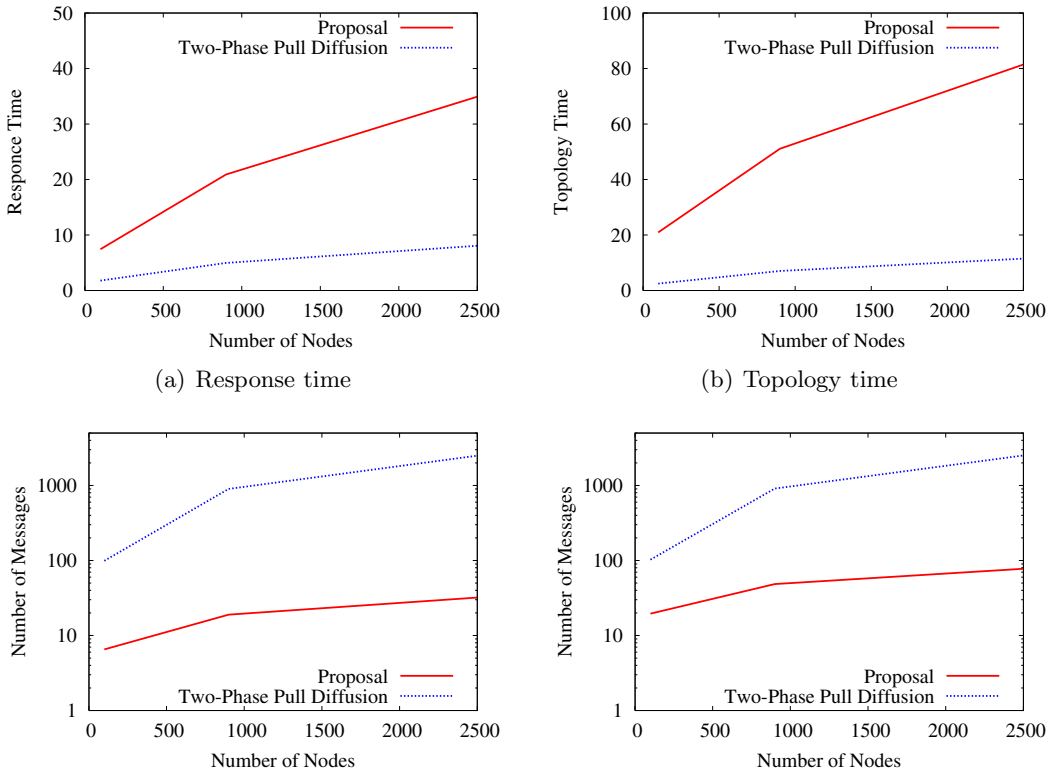
is established for information diffusion, ten sensor nodes are deployed at random locations in the wireless sensor network. As can be seen in the figure, newly added sensor nodes initially emit messages independently of their location. When a new sensor node receives a message from an existing sensor node, it sets the level value as the received value plus one. However, its timer has not been adjusted well yet. Therefore, the phase-lock condition of neighboring sensor nodes is lost as in the case of timer errors. However, as time passes, they begin to behave in synchrony with sensor nodes at the same distance from the core node and the phase-lock condition is re-established in the whole sensor network.

4.4.2 Effectiveness of the Mechanism

We next evaluate effectiveness of our communication mechanism. We consider wireless sensor networks of 100, 900, and 2500 sensor nodes randomly distributed in 10×10, 30×30, and 50×50 region, respectively. A core node is randomly chosen for data gathering or information diffusion. For comparison purposes, we also conduct simulation experiments for the directed diffusion [89-91] where per-hop delay is set at 0.1 time units. All results are averaged over 100 simulation experiments.

The response time indicates the duration from emission of an interest or a message with a new session identifier to reception of sensor information from all nodes. The topology time indicates the duration from emission of an interest or a message with a new session identifier to reception of reinforcement messages at all nodes or to establish the phase-lock condition. The number of messages indicates the average number of messages that a node sends and receives during the response time or the topology time. The data gathering ratio is defined as the ratio of data reached to a core node or a sink node to the number of nodes. The lifetime is defined as the duration from emission of an interest or a message with a new session identifier to death of any sensor node due to depletion of energy.

In Figs. 4.20(a) and 4.20(b), both of the response time and topology time with our mechanism are longer than those with the directed diffusion. A traveling wave is generated thorough local and mutual interactions, whereas the directed diffusion relies on message flooding. However, the overhead in terms of the number of messages is much smaller with our mechanism. It is only 1 to 6 % of the directed diffusion in the response time and 4 to 26 % in the topology time as shown in Figs. 4.20(c) and 4.20(d). Since a sensor node emits a message per cycle in our mechanism, the number of message increases in proportional to the response and topology time. As described in Section 4.2, the response time and



(a) Response time

(b) Topology time

(c) Number of messages in response time

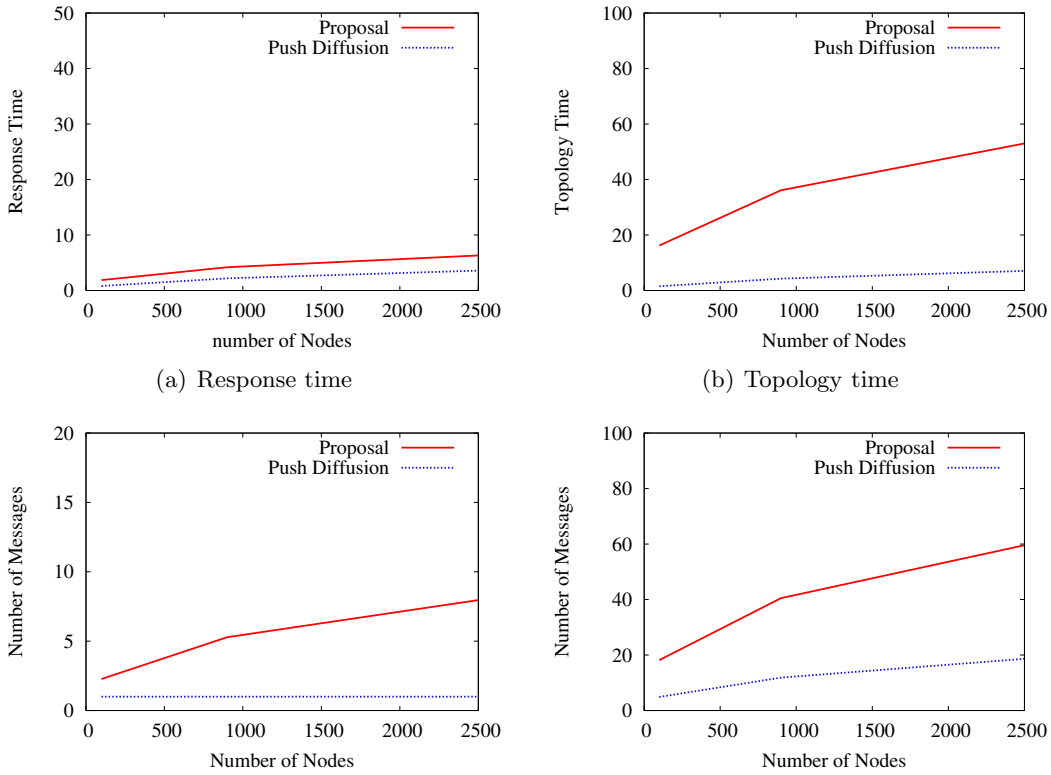
(d) Number of messages in topology time

Figure 4.20: Response time and topology time in gathering

topology time can be reduced by adjusting a PRC function and its parameters.

Figures 4.21(a) and 4.21(b) shows results for the case of information diffusion, where a randomly chosen node diffuses information to the whole sensor network. When comparing to the push diffusion of the directed diffusion, our proposal takes longer to diffuse information to all nodes. Differently from the data gathering scenario, the overhead is larger with our mechanism. It is 220 to 790 % of the directed diffusion in response time and 718 to 877 % in topology time as shown in Fig. 4.21(c) and 4.21(d). In the case of diffusion, only one source node floods exploratory data to all other nodes in the push diffusion, but our mechanism takes time to generate a traveling wave and thus requires much message exchanges.

Figure 4.22 shows the data gathering ratio against the packet loss probability in a 10×10 network. A sensor node randomly fails in transmitting a message at the packet loss probability shown on the x-axis. In Fig. 4.22, our mechanism always achieves higher data



(a) Response time

(b) Topology time

(c) Number of messages in response time

(d) Number of messages in topology time

Figure 4.21: Response time and topology time in diffusion

gathering ratio than the directed diffusion with the same packet loss probability. In our mechanism, broadcasting contributes to achieving multi-path effect and this leads to the higher gathering ratio.

Finally, we verify energy efficiency of our mechanism from a viewpoint of a lifetime of a sensor network of 100 nodes. As shown in Fig. 4.23(a), the lifetime with our mechanism is 1577 time units whereas that with the directed diffusion is 265 time units in the case of information gathering. Furthermore, by using a power-saving mode, the lifetime with our mechanism becomes as long as 2733 time units wheres that with the directed diffusion is 304 time units. On the contrary, as shown in Fig. 4.23(b), the lifetime with our mechanism is 1577 time units whereas that with the directed diffusion is 251 time units in the case of information diffusion. Furthermore, by using a power-saving mode, the lifetime with our mechanism becomes as long as 3680 time units whereas that with the directed diffusion is

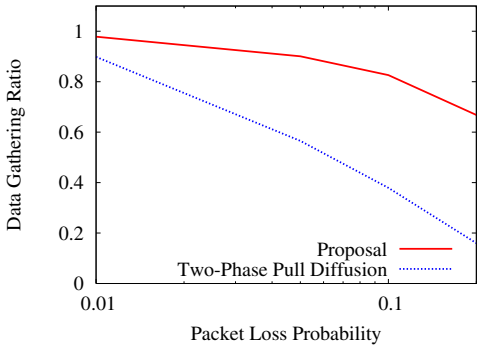
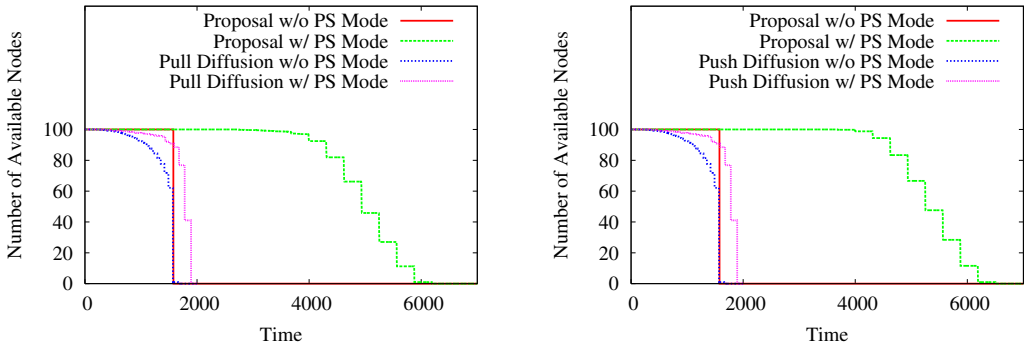


Figure 4.22: Data gathering ratio against packet loss probability



(a) Gathering (b) Diffusion

Figure 4.23: Number of available nodes

286 time units.

4.5 Implementation and Experimental Evaluation

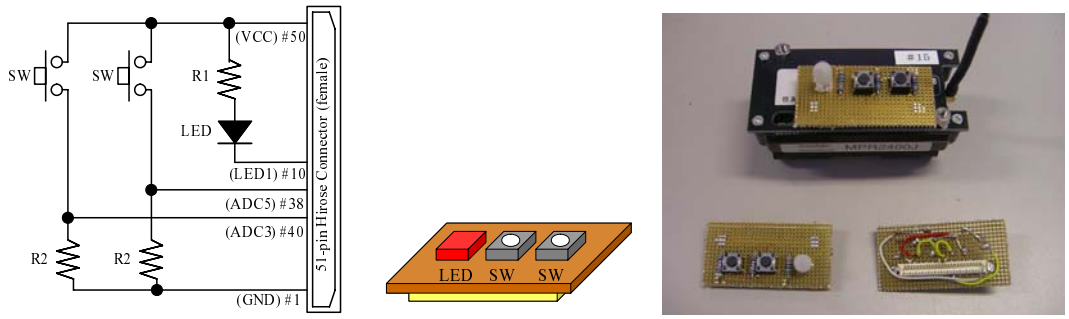
In this section, we describe our implementation of proposed mechanism on a real system, and evaluate our mechanism through practical experiments.

4.5.1 Implementation of the Mechanism

We implement our mechanism using a commercial sensor unit Crossbow MICAz [66]. It has an omni-directional antenna and employs B-MAC [93] and IEEE 802.15.4 [94] protocol

Table 4.1: Consumption current of MICAz

Module	Description	Consumption Current
MCU	ATMega128L (7.37 MHz, 8 bit)	12 mA (Active) 0.01 mA (Sleep)
Flash Memory	AT45DB014B (512 kB)	15 mA (Write) 4 mA (Read) 0.002 mA (Sleep)
Radio	CC2420 [95] (IEEE 802.15.4, 2.4 GHz ISM band)	19.7 mA (Receive) 8.5–17.4 mA (Transmit) 0.001 mA (Sleep)
Sensor	Light, Temperature, Acoustic, Magnetic Acceleration, Other	5 mA (Active) 0.005 mA (Sleep)



(a) Circuit Diagram of Attachment Module (b) Implemented Attachment Module

Figure 4.24: Diagram of attachment module for MICAz

on 2.4 GHz bandwidth for radio communication. The communication range varies from 20 to 100 m depending on the environment and the transmission power. It can be equipped with light, temperature, barometric pressure, acceleration/seismic, acoustic, magnetic, and other modules through Hirose DF-9 51 position interface.

Table 4.1 shows consumption current of MICAz. MICAz can reduce energy consumption by sleeping each modules. For example, when we use off-the-shelf battery with 2000 mAh of current capacity and we set data gathering cycle as 10 minutes and offset as 5 seconds, the lifetime of MICAz without sleep will be 2.26 days and the lifetime with sleep will be 132 days.

Table 4.2: List of parts for attachment module of MICAz

Mame	Model Number	Number
Connector	Hirose, DF9-51S-1V	1
5 mmϕ LED (Red)	Toshiba TLRH180P	1
5 mmϕ LED Diffusion Cap		1
1/4 W, 390 Ω Metal-film Resistor (R1)		1
1/4 W, 10 $k\Omega$ Metal-film Resistor (R2)		2
Tact Switch (SW)	Mitsumi SOA-113HS	2
1.27 mm Picth Board	Sunhayato ICB-028	1

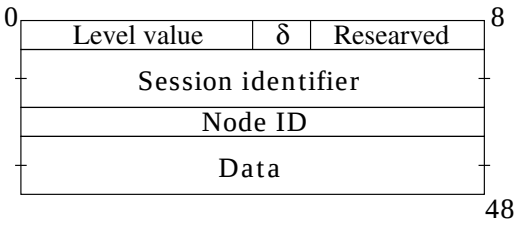


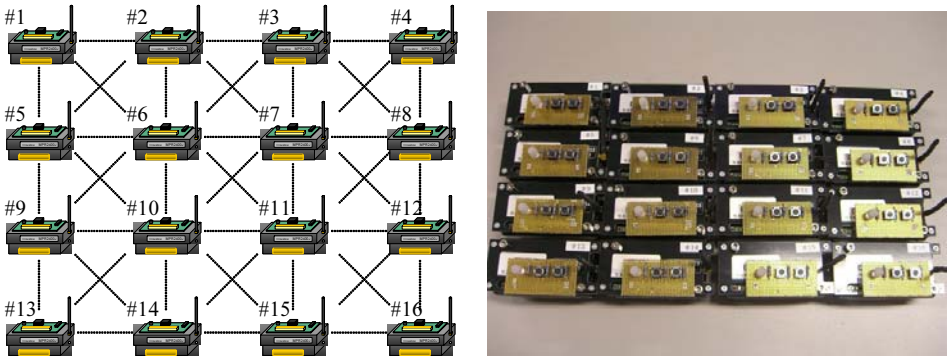
Figure 4.25: Packet format

To choose core node, we make attachment modules as shown in Fig. 4.24. An attachment module has a red LED and two push switches, and can be connected to MICAz through Hirose DF-9 51 position interface. The parts for the attachment module is listed in Table 4.2. The LED flashes when the sensor node broadcasts a message. A sensor node becomes a core node for information diffusion or gathering when its switch is pushed.

A timer is implemented by shifting phase ϕ_i by $0.1/T$ at every 100 milliseconds. A message is 48 bits long where the first 4 bits are for level value, 1 bit for δ, 3 bits reserved, 16 bits for session identifier, 8 bits for node id, and the last 16 bits for data as shown in Fig. 4.25.

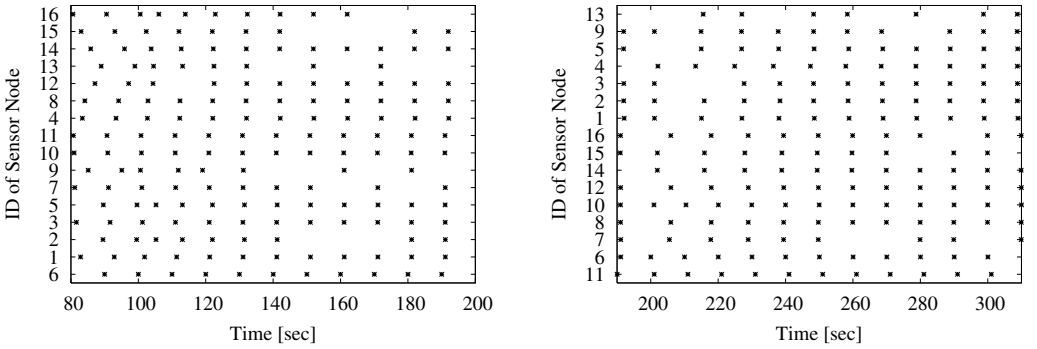
4.5.2 Experimental Evaluation

We confirm basic behavior of our mechanism on a sensor network consisting of 16 nodes arranged in a grid as shown in Fig. 4.26. To maintain the stable network topology, we introduce a filter with which a node ignores messages from non-neighboring nodes. A pair of nodes connected by a dotted line in Fig. 4.26 exchange messages. Since the filter is



(a) Relationship among sensor nodes (b) Photo of experiment

Figure 4.26: Experimental topology



(a) Timing of message emissions (diffusion) (b) Timing of message emissions (gathering)

Figure 4.27: Experimental evaluation of the mechanism

implemented on the application layer, collisions of radio signals among non-neighboring nodes occur. A cycle of data gathering or dissemination is set at 10 seconds. Other parameters and settings are the same as those used for the simulation experiments in the previous section. First, all sensor nodes periodically broadcast messages independently from each other. At time 100 seconds, sensor node 6 becomes a core node and initiates a data diffusion session. Then, at time 200 seconds, sensor node 11 initiates a new data session for data gathering.

Figures 4.27(a) and 4.27(b) show how the sensor network reached the phase-lock condition. Each mark stands for an instant when a sensor node emitted a message. For easier understanding, sensor nodes are sorted in order of the hop count from the core node. At first, all sensor nodes independently emit messages. However, by exchanging messages, the phase-lock condition for information diffusion eventually appears at about 130 seconds. Figure 4.27(a) shows that sensor nodes emit messages in order of the hop count from the core node, and thus information propagates from sensor node 6 to the edge of sensor network. From time 200, the phase-lock condition for information diffusion is first broken by initiating a new session. Then, the new phase-lock condition for information gathering appears at about 250 seconds, where information propagates from the edge of sensor network towards the sensor node 11.

As described above, we confirmed that our mechanism can gather or diffuse information in accordance with application requirements. However, due to collisions among synchronized packet broadcasts among sensor nodes of the same level, some packets are lost as shown blank in Fig. 4.27. In this experiment, data delivery ratio is about 87%. Packet loss leads to increase of delay and energy consumption.

4.5.3 Improvement and Evaluation of the Mechanism

To distribute timing of broadcast emissions, we improve the PRC function of sensor node i from Eq. (4.13) to following.

$$\Delta(\phi_i) \quad = \quad a \sin \frac{\pi}{g_i} \phi_i + b(g_i - \phi_i), \tag{4.14}$$

where $g_i = (1 - \delta_i \tau_i) \bmod 1$. Offset τ_i ($\tau_{min} < \tau_i \leq \tau_{max} = \tau$) is chosen randomly.

Figure 4.28 shows timing of message emissions for information gathering when Eq. (4.14) is used for PRC function. In Fig. 4.28, although node B and C have same level, node C first

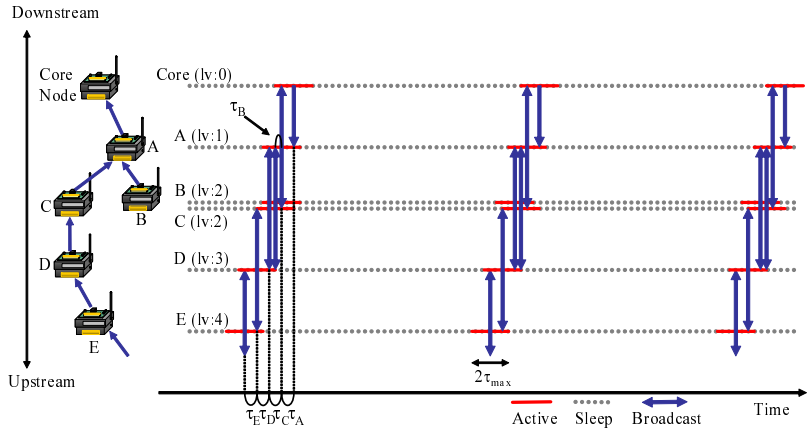
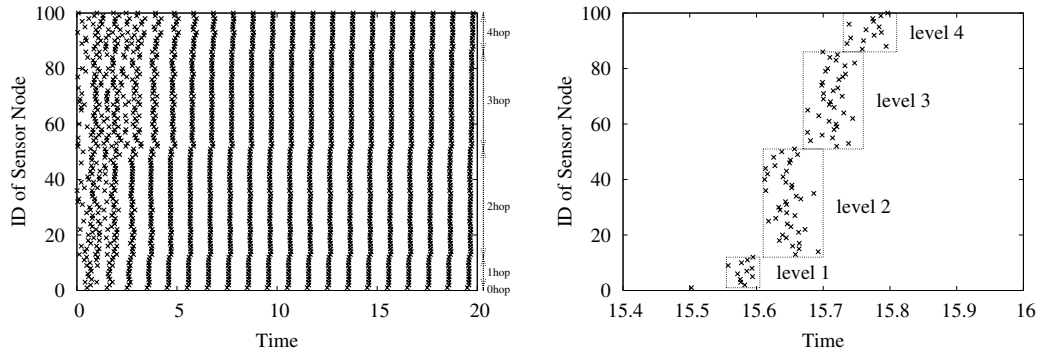


Figure 4.28: Timing of message emissions in the improved mechanism

broadcasts its message τ_C before node A broadcasts, and node B broadcasts its message τ_B before node A broadcasts.

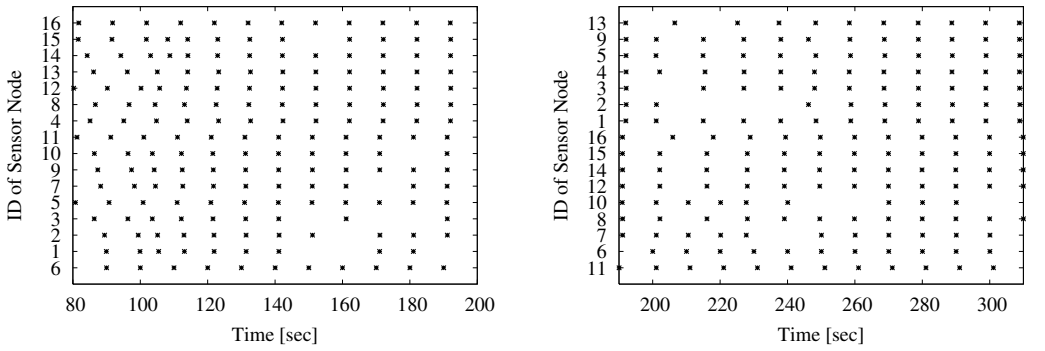
To confirm the basic behavior of the improved mechanism, we consider sensor networks of 100 sensor nodes randomly distributed in a 10×10 region as shown in Fig. 4.16. We use τ_{max} as 0.1, τ_{min} as 0.05, and same setting for other parameters as described in Section 4.4.1. From 0 to 20 time units, we chose a sensor node A as a core node for information diffusion. Figure 4.29 shows how the sensor network reached the phase-lock condition when the improved mechanism is used. Each mark stands for an instant when a sensor node emitted a message. Sensor nodes are sorted in order of the hop count from the core node. As shown in Fig. 4.29(a), by exchanging stimuli several times, the phase-lock condition is eventually accomplished and a regular pattern appears. Figure 4.29(b) shows that the improved mechanism can distribute timing of broadcast emission among sensor nodes of the same level.

We confirm effectiveness of the improved mechanism on a sensor network consisting of 16 nodes arranged in a grid as shown in Fig. 4.26. We use same settings as used in Section 4.5.2. Figures 4.30(a) and 4.30(b) show how the sensor network reached the phase-lock condition. Each mark stands for an instant when a sensor node emitted a message. As with the results in Section 4.5.2, by exchanging messages, the phase-lock condition for information diffusion eventually appears. Since the improved mechanism distributes timing of packet emission, data delivery ratio of about 95 % is accomplished.



(a) Timing of message emissions from 0 to 20 (b) Timing of message emissions from 15 to 16

Figure 4.29: Simulation evaluation of the improved mechanism



(a) Timing of message emissions (diffusion) (b) Timing of message emissions (gathering)

Figure 4.30: Experimental evaluation of the improved mechanism

4.6 Conclusion

In this chapter, we first investigated conditions that lead to a desired form of traveling wave regardless of the initial phase of oscillators in a PCO model. Then, we proposed a self-organizing communication mechanism in wireless sensor networks. Through simulation experiments, we confirmed that our scheme delivers sensor information to/from designated nodes in a more robust and energy-efficient manner than other methods. Our mechanism could extend the network lifetime by a factor of up to 12.8 compared to the directed diffusion method, which is a well-known reference model found in the lifetime. However, due

to the underlying biological model, we experienced a slight delay until the data dissemination process is completed. Furthermore, we implemented the mechanism using commercial wireless sensor units, MICAz. Since collisions among synchronized packet emissions affects the performance, we extended the mechanism to distribute timing of packet emission and consequently data delivery ratio of about 95 % was accomplished.

Chapter 5

A Data Gathering Mechanism Adaptive to Sensing Requirements for Wireless Sensor Networks

5.1 Introduction

In wireless sensor networks, sleep scheduling certainly plays an essential role in saving energy consumption for longer lifetime and guaranteeing communication delay for mission critical applications [27]. In Chapter 4, we considered energy-efficient scheduling for periodic communication involving all sensor nodes. In gathering type of our communication mechanism, message transmission first begins at the edge of a wireless sensor network. At the timing when those nodes at the furthest hop distance from the base station transmit messages, nodes which are closer to the base station by one hop, i.e. next hop nodes, are scheduled to wake up to receive the messages. After a while, they also transmit messages, and at this time, their next hop nodes are awake and ready to receive the messages. At the same time, the furthest nodes also receive the messages and go to sleep. As a consequence of such scheduling, we see concentric traveling waves of message propagation from the edge to the base station.

If the interval of data gathering is fixed, such sleep scheduling to wake up nodes from the edge of a wireless sensor network to a base station is effective in saving energy consumption and reducing data gathering delay. However, in some class of applications, a node need to

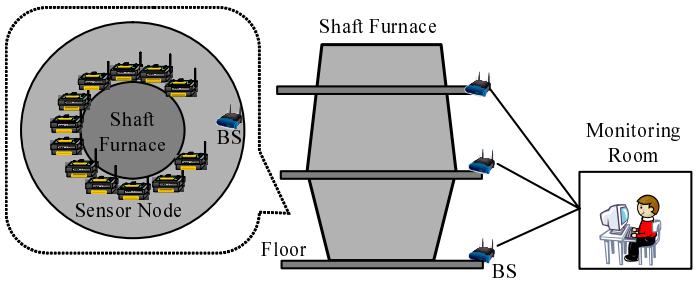


Figure 5.1: Monitoring of shaft furnace of steel plant

change its sensing frequency to monitor the region or target more frequently when it detects unusual condition and phenomena. Furthermore, the number of nodes which monitor and report the phenomena should be regulated in accordance with its criticality and importance. For example, we consider to deploy temperature and CO gas sensors on the surface of shaft furnace of steel plant as shown in Fig. 5.1. Temperature changes slowly in the order of hours and once increases, it stays high for a long period of time. Therefore, nodes are required to monitor temperature more frequently when temperature is changing, while they can decrease the sensing frequency under stable condition. On the other hand, CO gas suddenly appears and moves fast. Therefore, nodes are required to monitor CO gas more frequently than temperature while CO gas exists.

In this chapter, to tackle the above-mentioned problem, we propose a data gathering mechanism adaptive to dynamically changing sensing requirements. In our mechanism, nodes operate on traveling wave-based periodic data gathering at regular data gathering intervals. We organize traveling waves in a self-organizing fashion by adopting a pulse-coupled oscillator model [52] as described in Chapter 4. We assume that the regular sensing frequency is the same among sensors of different types. When condition of sensing target, such as temperature and gas concentration, changes at a certain point, surrounding nodes decide whether to monitor the target more frequently or not depending on the need for sensing. To regulate the number of nodes engaged in frequent monitoring of target in a self-organizing way, we adopt a response threshold model [69], i.e. a mathematical model of division of labor. Once a node considers to monitor the target more frequently, it changes its sensing frequency higher. So that obtained sensor data are forwarded to a base station at the higher frequency, nodes on the path to the base station also adapt to the new frequency. As a result, two or more traveling waves emerge in part or as a whole in a wireless sensor

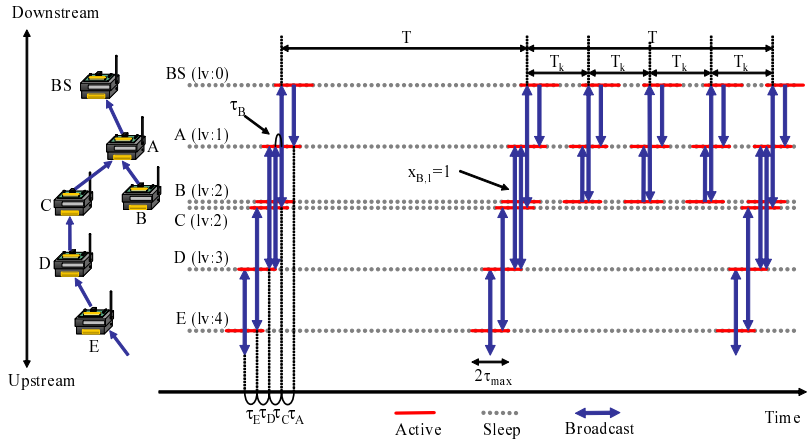


Figure 5.2: Broadcast timing of proposed mechanism

network.

The rest of this chapter is organized as follows. We propose a data gathering mechanism adaptive to sensing requirement in Section 5.2. Then, we show simulation results in Section 5.3. Finally, we conclude this chapter in Section 5.4.

5.2 A Data Gathering Mechanism Adaptive to Sensing Requirements

In this section, we propose a data gathering mechanism adaptive to sensing requirements. An example of behavior of our mechanism is illustrated in Fig. 5.2. In our mechanism, all nodes have k_{max} sensing devices, for example, temperature, light, and gas concentration, and monitors condition of k_{max} sensing targets. For example, a sensing target of a temperature sensor is temperature in the vicinity of a sensor, but we use sensor and sensing target interchangeably hereafter. Node i has two sensing states, *normal state* and *frequent state* for each sensor k $(1 \leq k \leq k_{max})$, and they are denoted as $x_{i,k} = 0$ and $x_{i,k} = 1$, respectively. Usually, node i is in *normal state* on all sensors, monitors sensing targets, and broadcasts sensor data at a regular interval of T seconds. When node i moves to *frequent state* on sensor k, it begins to operate at a new interval of $T_k = T/2^{c_k}$ seconds. To relay sensor data from node i to a base station, nodes on the path from node i to the base station also change their operation interval to T_k. Here, c_k $(0 \leq c_k \leq [\log_2(T/2\tau_{max})])$ is an integer

value pre-determined for sensing target k.

5.2.1 Adaptive Sensing using Response Threshold Model

To decide the sensing state for sensing target k at node i in a fully-distributed and self-organizing manner, we adopt the response threshold model of division of labor and task allocation in social insects [69]. There are several research work in application of the response threshold model to wireless sensor network control [96, 97].

Now, we consider demand $s_{i,k}$ of sensing target k at node i. Following the response threshold model, demand $s_{i,k}$ changes as

$$\frac{ds_{i,k}}{dt} = \delta_{i,k}(t) - \alpha \frac{m_{i,k}(t)}{n_{i,k}(t)}, \tag{5.1}$$

where $\delta_{i,k}(t)$ corresponds to the rate of increase and α determines the work efficiency. $m_{i,k}(t)$ and $n_{i,k}(t)$ represent the number of neighbor nodes performing frequent sensing on sensor k at time t and the number of neighbor nodes having sensor k, respectively. In the equation, when the increase of demand is equivalent to the work output, the demand does not change. If the number of workers are insufficient, the demand increases.

The definition of $\delta_{i,k}$ depends on application requirements. For example, in steel plant, temperature changes slowly in the order of hours and once increases, it stays high for a long period of time. Therefore, nodes are required to monitor temperature more frequently when temperature is changing, while they can decrease the sensing frequency under stable condition independently of the absolute value of temperature. It means that the rate of temperature change is appropriate as $\delta_{i,k}$ for temperature monitoring. On the other hand, CO gas suddenly appears and moves fast. The existence of CO gas is harmful to workers near the shaft. Therefore, the absolute value is more important for CO gas sensors and $\delta_{i,k}$ should be defined based on the value itself.

The probabilities that node i begins to work, i.e. monitor the sensing target more frequently and that node i stops frequent sensing are given by the following equations, respectively.

$$P(x_{i,k} = 0 \rightarrow x_{i,k} = 1) = \frac{s_{i,k}^2(t)}{s_{i,k}^2(t) + \theta_{i,k}^2(t)} \tag{5.2}$$

$$P(x_{i,k} = 1 \rightarrow x_{i,k} = 0) = p_{i,k}(t), \tag{5.3}$$

where $p_{i,k}(t) \in [0,1]$ is a parameter. For example, by using an inverse of changing rate of temperature as $p_{i,k}(t)$, we can expect that nodes immediately resume the normal sleep schedule once temperature becomes stable. The readiness of node to get engaged in a frequent sensing task depends on the threshold $\theta_{i,k}$. Threshold is adjusted depending on whether a node performs frequent sensing or not.

$$\frac{d\theta_{i,k}}{dt} = \begin{cases} -\xi, & \text{if } i \text{ performs frequent sensing} \\ \varphi, & \text{otherwise,} \end{cases} \tag{5.4}$$

where ξ and φ are parameters. The threshold adjustment makes specialists having a small threshold and keep sensing the target frequently. In addition, the threshold adjustment makes the response threshold model insensitive to parameter setting, such as the ragne of demand $s_{i,k}$ and α.

5.2.2 Data Gathering with Adaptive Intervals

Now, we explain how to accomplish adaptive data gathering on traveling wave-based communication. The mechanism is based on our previous mechanism described in Chapter 4.

Control Parameters

In our mechanism, node i maintains the phase $\phi_i \in [0,T]$ ($d\phi_i/dt = 1$), level value l_i, PRC (phase-response curve) function $\Delta_i(\phi_i)$, offset τ_i ($0 < \tau_{min} \leq \tau_i \leq \tau_{max} < 0.5T$), demand vector $\boldsymbol{S_i} = \{s_{i,k} | 1 \leq k \leq k_{max}\}$, threshold vector $\boldsymbol{\Theta_i} = \{\theta_{i,k} | 1 \leq k \leq k_{max}\}$, sensing state vector $\boldsymbol{X_i} = \{x_{i,k} | 1 \leq k \leq k_{max}\}$, relay flag vector $\boldsymbol{F_i} = \{f_{i,k} | 1 \leq k \leq k_{max}\}$, sensing state table $\boldsymbol{Y_i} = \{\forall j \boldsymbol{X_j}\}$ containing sensing state vectors of all neighbor nodes, and sensor data $\boldsymbol{D_i} = \{\boldsymbol{D_{i,k}} | 1 \leq k \leq k_{max}\}$. Entries of vectors are updated based on control information embedded in a received message from a neighbor node, which is emitted at the interval of sensing. It implies that there is no additional message transmission for control.

A level value l_i indicates the number of hops from a base station. Initially, a level value is set at infinity. A base station uses a level value of zero. The offset τ_i defines the interval of message emission between a node of level $l-1$ and that of level l. Offset τ_i is chosen randomly to avoid synchronous message emissions among nodes of the same level (see node B and C in Fig. 5.2). The maximum offset τ_{max} is determined taking into account the density of nodes. The PRC function determines the amount of phase shift on receiving a broadcast

message. To generate concentric traveling waves of message propagation regardless of the initial phase condition, we use the following PRC function for all nodes as described in Chapter 4.

$$\Delta_i(\phi_i) = a \sin \frac{\pi}{\tau_i}\phi_i + b(\tau_i - \phi_i), \tag{5.5}$$

where a and b are parameters which determine the speed of convergence. A relay flag $f_{i,k}$ is used to notify downstream nodes of the existence of upstream nodes in *frequent state*. When sensor node i relays data to downstream nodes from upstream nodes in *frequent state*, the relay flag $f_{i,k}$ is set at one, otherwise, $f_{i,k}$ is set at zero. Sensor data $\boldsymbol{D_{i,k}}$ contains all sensor data, together with originator's id, generated or received during the wake-up period for sensing target k.

Node Behavior

Node i behaves in accordance with its phase ϕ_i and relay flag vector $\boldsymbol{F_i}$. Node i wakes up when $\phi_i \mod \min_k T_k$ becomes $\min_k T_k - \tau_{max}$, where $\min_k T_k = T/\max_k(f_{i,k}2^{c_k}, 1)$. In Fig. 5.2, node A with sensor $k = 1$ additionally wakes up three times between regular sensing of interval of T, for having $\min_k T_k = T/4$ with $f_{A,1} = 1$, for example. Upstream nodes setting the same relay flag (node B in Fig. 5.2) are scheduled to broadcast a message during $\min_k T_k - \tau_{max}$ and $\min_k T_k$ from the node i's viewpoint. After waking up, node i initializes its relay flag vector $\boldsymbol{F_i}$ to all zero, clears sensor data $\boldsymbol{D_i}$, and waits for message reception.

When node i receives a message from upstream node j whose level value must be $l_j = l_i + 1$, node i deposits the received sensor data $\boldsymbol{D_j}$. The broadcast message also contains relay flag vector $\boldsymbol{F_j}$, sensing state vector $\boldsymbol{X_j}$. If any of relay flag $f_{j,k} \in \boldsymbol{F_j}$ is set at one in the received message, node i sets corresponding relay flag $f_{i,k}$ at one. In addition, node i replaces or adds the entry of node j in its sensing state table $\boldsymbol{Y_i}$ by the received sensing state vector $\boldsymbol{X_j}$.

When node i receives a message from node j with $l_j = l_i$, it checks whether the received sensor data $\boldsymbol{D_j}$ covers its own sensor data $\boldsymbol{D_i}$. Those sensor data contained in the broadcast message of node j are removed from $\boldsymbol{D_i}$. If $\boldsymbol{D_{i,k}}$ becomes empty, node i sets relay flag $f_{i,k}$ at zero. As a result, the amount of sensor data in a message can be reduced and the number of sensor nodes involved in frequent relaying could be further reduced as shown in Fig. 5.3.

Then, when $\phi_i \mod \min_k T_k$ reaches $\min_k T_k - \epsilon$, node i determines whether it moves

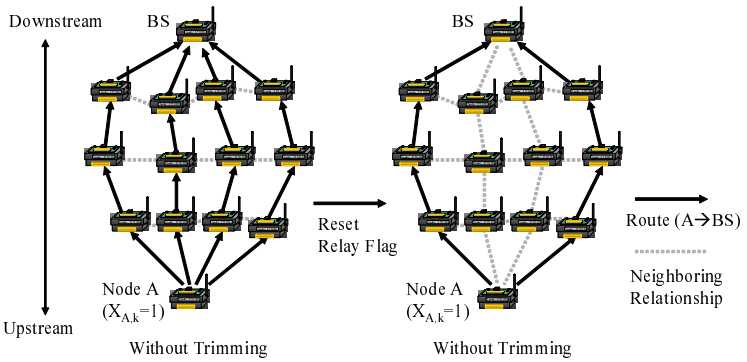


Figure 5.3: An example of message reduction

to *frequent state* for sensing target $\forall k \in \{k|\phi_i \mod T_k = T_k - \epsilon\}$ or not. ϵ $(0 < \epsilon < \tau_{min})$ corresponds to the sensing delay. First, node i calculates $n_{i,k}$ and $m_{i,k}$ from sensing state table $\boldsymbol{Y_i}$. Next, node i derives a demand vector $\boldsymbol{S}_i$ using Eq. (5.1). Then, node i determines its sensing state vector $\boldsymbol{X_i}$ using Eq. (5.2) and Eq. (5.3). If any of sensing state $x_{i,k}$ is set at one, node i monitors sensing target k, deposits sensor data, and sets its relay flag $f_{i,k}$ at one. If ϕ_i is $T - \epsilon$, i.e. timing for regular sensing, node i monitors all sensing targets and deposits sensor data. After that, node i adjusts its threshold vector $\boldsymbol{\Theta}_i$ using Eq. (5.4).

When $\phi_i \mod \min_k T_k$ becomes zero in $0 < \phi_i < T$, or ϕ_i reaches T, node i broadcasts a message, which is received by any of awake nodes in the range of radio communication. A message that node i emits contains node id i, level value l_i, sensing state vector $\boldsymbol{X_i}$, relay flag vector $\boldsymbol{F_i}$, synchronization flag z_i, and sensor data $\boldsymbol{D_i}$. z_i is set at one only if the phase ϕ_i is T on broadcasting the message. After broadcasting, the phase reaching T goes back to zero.

After the broadcasting, it keeps awake for τ_{max}. When node i receives a message having synchronization flag z_j of one from node j with $l_j < l_i$, it sets its level value l_i at $l_j + 1$ and shifts its phase by an amount $\Delta_i(\phi_i)$ in Eq. (5.5). The phase shift is done only once during the τ_{max} awake period. When node i receives a message from node j with $l_j = l_i$, it only updates sensing state table $\boldsymbol{Y_i}$. After τ_{max} later from its broadcasting, node i goes to sleep. Therefore, a node is awake for the duration of $2\tau_{max}$ in one data gathering interval $\min_k T_k$, i.e. duty cycle becomes $2\tau_{max}/\min_k T_k$.

5.2.3 Overhead of the Mechanism

Now, let us consider protocol overhead and complexity. Each sensor node operates on the phase of timer and wakes up at the frequency of sensing. A message containing sensor data and control information is broadcast once per awake period.

Information that node i has to maintain are, level value l_i, demand vector $\boldsymbol{S_i}$, threshold vector $\boldsymbol{\Theta_i}$, sensing state vector $\boldsymbol{X_i}$, relay flag vector $\boldsymbol{F_i}$, and sensing state table $\boldsymbol{Y_i}$. Then, we can formulate the total amount as, $|l_i| + |\boldsymbol{S_i}| + |\boldsymbol{\Theta_i}| + |\boldsymbol{X_i}| + |\boldsymbol{F_i}| + |\boldsymbol{Y_i}| + |\boldsymbol{D_i}|$. When we assume that a level value needs 4 bits (up to 15 hops), each of demand and threshold value is expressed by 32 bits, each of relay flag and sensing state needs 1 bit, and the number of neighbors is n, the total amount becomes $4 + (66 + n)k_{max}$ bits. In our scenario in steel plant, $k_{max} = 2$, i.e. temperature and CO gas concentration, and the number of neighbors is a few dozen. Therefore, memory size required for control information is less than 200 bits. Most of off-the-shelf sensor nodes can afford this amount. For example, MICAz [66] has 32 kbits of memory.

Control information in a message includes node id i, level value l_i, sensing state vector $\boldsymbol{X_i}$, relay flag vector $\boldsymbol{F_i}$, and synchronization flag z_i. Then, we can formulate the total amount as, $|i| + |l_i| + |\boldsymbol{X_i}| + |\boldsymbol{F_i}| + |z_i|$. When we assume a node id requires 10 bits (1023 nodes) and a synchronization flag requires 1 bit, the total amount becomes $15 + 2k_{max}$ bits. If $k_{max} = 2$, only 19 bits are needed for control information in a message. For example, a MICAz [66] sensor node with TinyOS platform uses 144 bits for total size of control information, i.e. header and meta-data field, as default [98].

5.3 Simulation Experiments

In this section, we show results of simulation experiments. We consider a wireless sensor network of 200 nodes randomly distributed in 100 m $\times$ 100 m region as shown in Fig. 5.4. A base station is located in the center of the region. Both sensing and communication ranges are fixed at 20 m. We consider two sensing targets, i.e. temperature and CO gas concentration. The normal data gathering interval is $T = 160$ seconds. For unusual condition, we set $c_{temp} = 2$ and $c_{gas} = 4$, and therefore, $T_{temp} = 40$ seconds and $T_{gas} = 10$ seconds, respectively.

We use $\delta_{i,temp} = \beta|dv_{i,temp}/dt|$ and $\delta_{i,gas} = v_{i,gas}$ in Eq. (5.1), and $p_{i,temp} = 1 - \beta|dv_{i,temp}/dt|$ and $p_{i,gas} = 1 - v_{i,gas}$ in Eq. (5.3) for all nodes. $v_{i,temp} \in [0, 1]$ and $v_{i,gas} \in [0, 1]$

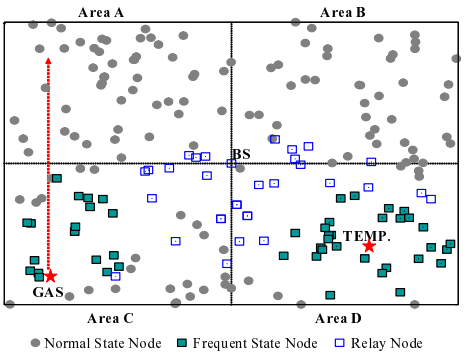


Figure 5.4: Node distribution and snapshot at 1200 seconds in simulation experiments

are temperature and CO gas concentration normalized by its maximum value, respectively. We use $\beta = 1000$. A node consumes 52.2 mW for transmission, 59.1 mW for reception, 60 μW in an idle mode, and 3 μW in a sleep mode [66]. τ_{max} and τ_{min} is set at 1 second and 0.5 second, respectively. We assume that control information in a message amounts to 24 bits and one sensor data amounts to 16 bits. We use $a = 0.01$ and $b = 0.5$ for PRC in Eq. (5.5) and the parameters of response threshold model are set to $\alpha = 1$, $\xi = 0.01$, and $\varphi = 0.001$.

Until 500 seconds, nothing happens where temperature v_{temp} is 0.1 and no CO gas is observed. From 500 seconds, temperature begins to linearly increase at a randomly chosen location in the area D. At 1500 seconds, temperature at the location reaches 0.9 and stays high until the end of simulation. During the temperature increase, CO gas of concentration v_{gas}=0.8 leaks out from a randomly chosen location in the area C at 1000 seconds. The gas moves to the area A at the speed of 0.08 m/s. At 2000 seconds, the gas disappears.

Figure 5.5 illustrates the behavior of nodes. Each mark corresponds to the instant when a node, whose id is shown on the left y-axis, broadcasts a message. Nodes are numbered based on the location as shown on the right y-axis. Figures 5.6(a) and 5.6(b) show the number of nodes in *frequent state* for each of sensing targets. At first, all nodes are in the *normal state*. At the end of awake period at 500 seconds, some nodes in the area D decide to monitor and report temperature more often to answer the increased sensing demand caused by temperature change. From the next timing, at most 160 seconds later, they begin frequent sensing of temperature as shown in Fig. 5.6(a). Since they set relay flag $f_{i,temp}$ in broadcast messages, nodes on the path to the base station also set their relay flag

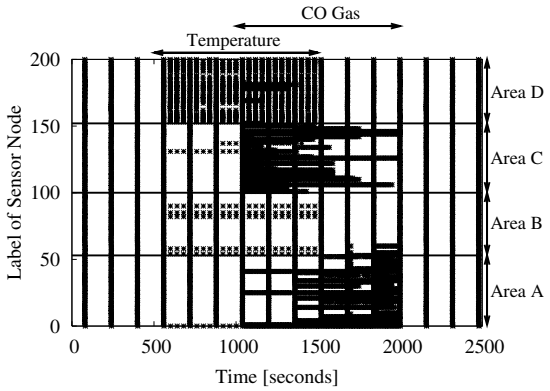
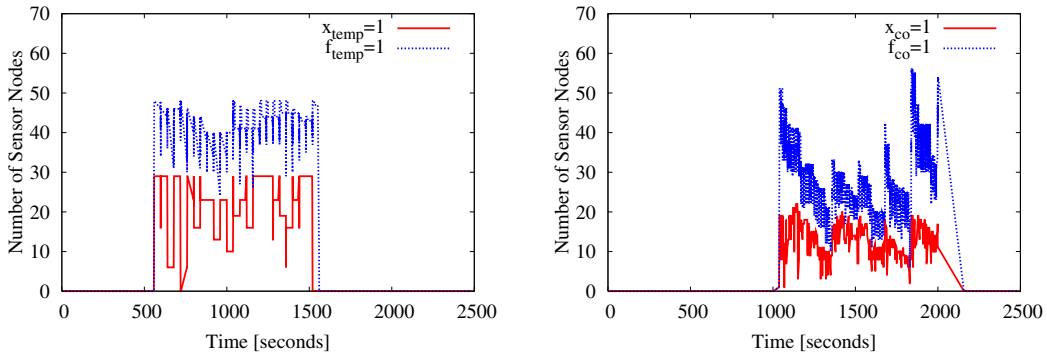


Figure 5.5: Timing of message emissions



(a) Temperature

(b) CO gas

Figure 5.6: Number of nodes in frequent state

and begin to operate at the shorter intervals of $T_{temp} = 40$ seconds. At 1500 seconds, nodes find that temperature becomes stable, and they go back to *normal state*.

Figure 5.4 illustrates snapshot of the wireless sensor network at 1200 seconds. Each filled circle, filled square, and open square correspond to a node in the *normal state*, a node in the *frequent state*, and a relay node, respectively. Some nodes between the base station and nodes in the *frequent state* stay in the normal state and keep operating at the normal interval. It contributes reduction of energy consumption and probability of collisions among nodes.

In a similar manner, at 1000 seconds, some nodes in the area C begin to operate at the shorter intervals of $T_{gas} = 10$ seconds for CO gas leakage. As CO gas moves to the area

A, nodes in the area C eventually go back to *normal state* while nodes in the area A move to the *frequent state* as shown in Fig. 5.5. Nodes in the frequent state evaluates Eqs. (5.2) and (5.3) often whereas nodes moving from the normal state to the frequent state begins frequent sensing at the next wake-up. Therefore, we observe the gradual decrease in short time scale and the sudden increase in long time scale in Fig. 5.6(b).

During simulation experiments, total energy consumption with our proposed mechanism is 57.6 mJ per node and duty cycle is 0.03 per node. On the other hand, when we adopt 10 seconds as the regular data gathering interval to prepare for CO gas leakage, the per-node energy consumption becomes 632 mJ and duty cycle is 0.2. Therefore, we can conclude that our mechanism accomplishes autonomous and energy-efficient data gathering satisfying dynamically changing sensing requirements.

5.4 Conclusion

In this chapter, we considered adaptation of intervals of sensing and data gathering period, since the interval depends on application requirements and surrounding conditions in wireless sensor networks. We proposed a data gathering mechanism adaptive to sensing requirements in wireless sensor networks composed nodes with multiple sensing capabilities. To accomplish self-organizing control, we adopted the response threshold model for adaptive sensing task engagement and the pulse-coupled oscillator model for energy-efficient transmission and sleep scheduling. Through simulation experiments, we confirmed that autonomous and energy-efficient data gathering can be accomplished satisfying dynamically changing sensing requirements in a more energy-efficient manner.

Chapter 6

Conclusion

Communication mechanisms for future cooperative information networks must cope with the heterogeneity of connected devices and dynamic changes in the network environment. In this thesis, we considered self-adaptive communication mechanisms for cooperative information networks focusing on two very different types of information networks, i.e. video streaming systems and wireless sensor networks.

First, we considered a video streaming system. We introduced proxy caching servers to the video streaming system to achieve scalable, low-delay, and high-quality video streaming. By cooperation among proxies, the video streaming system can reduce the perceived network latency and achieve a higher degree of content availability. Furthermore, we provided capability of video quality adaptation, i.e. video filters, at the proxy to handle heterogeneous and dynamically changing requirements on the quality of the video data provided to heterogeneous users in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality. We began this work by proposing, designing, and implementing a proxy caching system for MPEG-4 video streaming services by using common applications for server and client programs. Then, we extended our caching mechanism by considering cooperation among the proxies. Through simulation and experimental evaluations, it was shown that our proposed mechanism can provide users with continuous and high-quality video streaming services under dynamically changing network conditions in comparison with independent and non-adaptive caching mechanisms.

Next, we considered wireless sensor networks which should be operated in an energy-efficient, adaptive, robust, fully-distributed, and self-organizing manner. We proposed two self-adaptive communication mechanisms by adopting bio-inspired models to handle various

and dynamically changing application requirements on communication patterns, sensing frequencies, and the number of nodes to monitor and report the phenomena. The first communication mechanism adapts to various types of communication, i.e. diffusion and gathering, in accordance with the application requirements in wireless sensor networks. We investigated conditions that lead to a desired form of traveling wave regardless of the initial phases in a pulse-coupled oscillator model, and proposed a self-adaptive and self-organizing communication mechanism. Through simulation experiments, we confirmed that our mechanism delivers sensor information to/from designated nodes in a more energy-efficient and robust manner than other methods, at the slight cost of increased time to generate a traveling wave. Implementations using commercial MICAz sensor units gave further evidence of the good applicability of our approach. Then, we extended the first self-adaptive communication mechanism by considering the adaptation of sensing frequencies and the number of nodes which monitor and report the phenomena. We proposed a self-adaptive data gathering mechanism in wireless sensor networks composed of nodes with multiple sensing capabilities by adopting the response threshold model for adaptive sensing task engagement. Through simulation experiments, we confirmed that autonomous and energy-efficient data gathering can be accomplished, satisfying the dynamically changing sensing requirements.

From above findings, for very distinct types of two information networks, i.e. video streaming systems and wireless sensor networks, we can conclude that self-adaptability is a key feature to accomplish future cooperative information networks which suffer very much from unpredictable, unreliable, dynamically chainging, and selfish behavior of heterogeneous users and devices as well as the external and environmental conditions. Although a lot of problems are left as open issues to be solved, we believe that those discussions and results in this thesis would provide insights toward a future information society.

Bibliography

[1] N. Niebert, A. Schieder, J. Zander, and R. Hancock, *Ambient Networks: Co-Operative Mobile Networking for the Wireless World*. Wiley, 2007.

[2] Q. H. Mahmoud, *Cognitive Networks: Towards Self-Aware Networks*. Wiley, 2007.

[3] N. Wakamiya, K. Leibnitz, and M. Murata, "Noise-assisted control in information networks," in *Proceedings of Frontiers in the Convergence of Bioscience and Information Technologies (FBIT 2007)*, pp. 11–13, Oct. 2007.

[4] D. B. Johnson, "Routing in ad hoc networks of mobile hosts," in *Proceedings of the Workshop in Mobile Computing Systems and Applications*, pp. 158–163, Dec. 1994.

[5] C. Perkins and E. Royer, "Ad hoc on-demand distance vector routing," in *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.

[6] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C.Gill, "Integrated coverage and connectivity configuration in wireless sensor networks," in *Proceedings of ACM 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pp. 28–39, Nov. 2003.

[7] F. Dressler, *Self-Organization in Sensor and Actor Networks*. Wiley, 2007.

[8] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artifical Systems*. Oxford Univ Pr., 1999.

[9] M. Murata, "Biologically inspired communicaiton network control," in *Proceedings of SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, May 2004.

[10] K. Leibnitz, N. Wakamiya, and M. Murata, "Self-adaptive ad-hoc/sensor network routing with attractor-selection," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM 2006)*, pp. 1–5, Nov. 2006.

[11] K. Leibnitz, N. Wakamiya, and M. Murata, "Biologically inspired self-adaptive multipath routing in overlay networks," *Communication of the ACM Special Issue on Self-managed Systems and Services*, vol. 49, pp. 62–67, Mar. 2006.

[12] G. D. Caro, F. Ducatelle, and L. M. Gambardella, "AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks," *European Transactions on Telecommunications*, vol. 16, pp. 443–455, Oct. 2005.

[13] "YouTube." `http://www.youtube.com/`.

[14] "Google Video." `http://video.google.com/`.

[15] "GyaO." `http://www.gyao.jp/`.

[16] J. Liu and J. Xu, "Proxy caching for media steaming over the Internet," *IEEE Communication Magazine*, pp. 88–94, Aug. 2004.

[17] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal proxy cache allocation for efficient streaming media distribution," *IEEE Transactions on Multimedia*, vol. 6, pp. 366–374, Apr. 2004.

[18] J. Song, "Segment-based proxy caching for distributed cooperative media content servers," *ACM SIGOPS Operating Systems Review*, vol. 39, pp. 22–33, Jan. 2005.

[19] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "DISC: dynamic interleaved segment caching for interactive streaming accesses," in *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pp. 763–772, June 2005.

[20] A. T. S. Ip, J. Liu, and J. C.-S. Lui, "COPACC: An architecture of cooperative proxy-client caching system for on-demand mediea streaming," *IEEE Transaction on Parallell and Distributed Systems*, vol. 18, pp. 70–83, Jan. 2007.

[21] B. Shen, S.-J. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Transactions on Multimedia*, vol. 6, pp. 375–386, Apr. 2004.

[22] M. Zink, J. Schmitt, and C. Griwodz, "Layer-enabled video streaming: a proxy's perspective," *IEEE Communication Magazine*, pp. 96–103, Aug. 2004.

[23] C.-L. Lin, H.-H. Lee, C.-L. Chan, and J.-S. Wang, "Cooperative proxy framework for layered video streaming," in *Proceedings of the 2005 IEEE Global Telecommunications Conference (GLOBECOM 2005)*, vol. 1, Nov. 2005.

[24] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, Mar. 2002.

[25] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE Networks*, vol. 18, pp. 45–50, July 2004.

[26] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, pp. 325–349, May 2005.

[27] L. Wang and Y. Xiao, "A survey of energy-efficient scheduling mechanisms in sensor networks," *Mobile Networks and Applications*, vol. 11, pp. 723–740, Oct. 2006.

[28] C. F. Garcia-Hernández, P. H. Ibrargüengoytia-González, J. García-Hernández, and J. A. Pérez-Díaz, "Wireless sensor networks and applications: a survey," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 7, pp. 264–273, Mar. 2007.

[29] M. Younis and K. Akkaya, "Strategies and techniques for node placement in wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 6, pp. 621–655, June 2008.

[30] K. L. Mills, "A brief survey of self-organization in wireless sensor networks," *Wireless Communicaitons and Mobile Computing*, vol. 7, pp. 823–834, Sept. 2007.

[31] Y. Taniguchi, A. Ueoka, N. Wakamiya, M. Murata, and F. Noda, "Implementation and evaluation of proxy caching system for MPEG-4 video streaming with quality adjustment mechanism," *Technical Report of IEICE (NS2003-45)*, pp. 45–48, June 2003. (in Japanese).

[32] Y. Taniguchi, A. Ueoka, N. Wakamiya, M. Murata, and F. Noda, "Implementation and evaluation of proxy caching system for MPEG-4 video streaming with quality adjustment mechanism," in *Proceedings of The 5th AEARU Workshop on Web Technology*, pp. 27–34, Oct. 2003.

[33] Y. Taniguchi, N. Wakamiya, and M. Murata, "A proxy caching system for MPEG-4 video streaming with a quality adaptation mechanism," *WSEAS Transactions on Communications*, vol. 6, pp. 824–832, Oct. 2007.

[34] Internet Streaming Media Alliance, "Internet streaming media alliance implementation specification version 1.0," Aug. 2001.

[35] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," *Internet Request for Comments 3448*, Jan. 2003.

[36] N. Wakamiya, M. Murata, and H. Miyahara, "Cooperative video streaming mechanisms with video quality adjustment," in *Proceedings of the 4th Asia-Pasific Symposium on Information and Telecommunication Technologies (APSITT 2001)*, pp. 106–110, Nov. 2001.

[37] N. Wakamiya, M. Murata, and H. Miyahara, "Video streaming systems with cooperative caching mechanisms," in *Proceedings of SPIE International Symposium ITCom 2002*, pp. 305–314, July 2002.

[38] N. Wakamiya, M. Murata, and H. Miyahara, "On proxy-caching mechanisms for cooperative video streaming in heterogeneous environment," in *Proceedings of the 5th IFIP/IEEE Ineternational Conference on Management of Multimedia and Mobile Networks and Services (MMNS 2002)*, pp. 127–139, Oct. 2002.

[39] Y. Taniguchi, N. Wakamiya, and M. Murata, "Implementation and evaluation of cooperative proxy caching system for video streaming services," *Technical Report of IEICE (IN2003-190)*, pp. 13–18, Feb. 2004. (in Japanese).

[40] Y. Taniguchi, N. Wakamiya, and M. Murata, "Implementation and evaluation of cooperative proxy caching mechanisms for video streaming services," in *Proceedings of SPIE's International Symposium on the Convergence of Information Technologies and Communications (ITCom 2004)*, pp. 288–299, Oct. 2004.

[41] Y. Taniguchi, N. Wakamiya, and M. Murata, "Quality-aware cooperative proxy caching for video streaming services," submitted to *Journal of Networks*, Apr. 2008.

[42] Y. Taniguchi, N. Wakamiya, and M. Murata, "A distributed and self-organizing data gathering scheme in wireless sensor networks," in *Proceedings of the 6th Asia-Pacific*

Symposium on Information and Telecommunication Technologies (APSITT 2005), pp. 299–304, Nov. 2005.

[43] Y. Taniguchi, N. Wakamiya, and M. Murata, "A distributed and self-organizng communication mechanism based on traveling wave phenomena for wireless sensor networks," *Technical Report of IEICE (NS2006-48)*, pp. 17–20, July 2006. (in Japanese).

[44] Y. Taniguchi, N. Wakamiya, and M. Murata, "A self-organizing communication mechanism using traveling wave phenomena for wireless sensor networks," in *Proceedings of the 2nd International Workshop on Ad Hoc, Sensor and P2P Networks (AHSP 2007)*, pp. 562–569, Mar. 2007.

[45] Y. Taniguchi, N. Wakamiya, and M. Murata, "Implementaiton and evaluation of traveling wave based communication mechanism for wireless sensor networks," *Technical Report of IEICE (NS2007-40)*, pp. 1–6, July 2007. (in Japanese).

[46] Y. Taniguchi, N. Wakamiya, and M. Murata, "A communication mechanism using traveling wave phenomena for wireless sensor networks," in *Proceedings of the 1st Annual IEEE International Workshop: From Theory to Practice in Wireless Sensor Networks (t2pWSN 2007)*, May 2007.

[47] Y. Taniguchi, N. Wakamiya, and M. Murata, "A traveling wave based communication mechanism for wireless sensor networks," *Journal of Networks*, vol. 2, pp. 24–32, Sept. 2007.

[48] Y. Taniguchi, N. Wakamiya, and M. Murata, "Demo abstract: a traveling wave-based self-organizing communication mechanism for WSNs," in *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys 2007)*, pp. 399–400, Nov. 2007.

[49] B. Ermentrout and J. Rinzel, "Beyond a pacemaker's entrainment limit: phase walkthrough," *The American Journal of Physiology - Regulatory, Integrative and Comparative Physiology*, pp. 102–106, 1984.

[50] R. E. Mirollo and S. H. Strogatz, "Synchronization of pulse-coupled biological oscillators," *Society for Industrial and Applied Mathematics Journal*, vol. 50, pp. 1645–1662, Dec. 1990.

[51] B. Ermentrout, "An adaptive model for synchrony in the firefly Pteroptyx malaccae," *Journal of Mathematical Biology*, pp. 571–585, 1991.

[52] P. Goel and B. Ermentrout, "Synchrony, stability, and firing patterns in pulse-coupled oscillators," *Physica D*, pp. 191–216, Mar. 2002.

[53] H.-A. Tanaka, "Computer simulation of population synchrony for P. effulgens," *Insects and Nature*, June 2004.

[54] Y.-W. Hong and A. Scaglione, "A scalable synchronization protocol for large scale sensor networks and its appliations," *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 1085–1099, May 2005.

[55] S. Barbarossa, "Self-organizing sensor networks with information propagation based on mutual coupling of dynamic systems," in *Proceedings of the 2005 International Workshop on Wireless Ad-hoc Networks (IWWAN 2005)*, May 2005.

[56] S. F. Bush, "Low-energy network time synchronization as an emergent property," in *Proceedings of the 14th International Conference on Computer Communications and Networks (ICCCN 2005)*, pp. 93–98, Oct. 2005.

[57] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, pp. 142–153, Nov. 2005.

[58] A.-S. Hu and S. D. Servetto, "On the scalability of cooperative time synchronization in pulse-connected networks," *IEEE Transactions on Information Theory*, vol. 52, pp. 2725–2748, June 2006.

[59] A. Tyrrell, G. Auer, and C. Bettstetter, "Synchronization inspired from nature for wireless meshed networks," in *Proceedings of the 2nd IEEE International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2006)*, pp. 1–4, Sept. 2006.

[60] O. Simeone and U. Spagnolini, "Distributed synchronization for wireless sensor networks with coupled discrete-time oscillators," *EURASIP Journal on Wireless Communication and Networking Special Issue on Novel Techniques for Analysis & Design of Cross-Layer Optimized Wireless Sensor Networks*, vol. 2007, 2007.

[61] J. Degesys, I. Rose, A. Patel, and R. Nagpal, "DESYNC: self-organizing desynchronization and TDMA on wireless sensor networks," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (ISPN 2007)*, pp. 11–20, Apr. 2007.

[62] A. Patel, J. Degesys, and R. Nagpal, "Desynchronization: the theory of self-organizng algorithms for round-robin scheduling," in *Proceedings of the 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pp. 87–96, July 2007.

[63] J. Degesys, P. Basu, and J. Redi, "Synchronization of strongly pulse-coupled oscillators with refractory periods and random medium access," in *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008)*, pp. 1976–1980, Mar. 2008.

[64] N. Wakamiya and M. Murata, "Synchronization-based data gathering scheme for sensor networks," *IEICE Transactions on Communications*, vol. E88-B, pp. 873–881, Mar. 2005.

[65] S. Kashihara, N. Wakamiya, and M. Murata, "Implementation and evaluation of a synchronization-based data gathering scheme for sensor networks," in *Proceedings of the IEEE International Conference on Communication, Wireless Networking (ICC 2005)*, pp. 3037–3043, May 2005.

[66] "MOTE." `http://www.xbow.com/Products/wproductsoverview.aspx`.

[67] Y. Taniguchi, N. Wakamiya, M. Murata, and T. Fukushima, "An autonomous data gathering scheme adaptive to sensing requirements for industrial environment monitoring," submitted to *the 2nd International Conference on New Technologies, Mobility and Security (NTMS 2008)*, June 2008.

[68] Y. Taniguchi, N. Wakamiya, M. Murata, and T. Fukushima, "A traveling wave based data gathering scheme adaptive to sensing requirements," to be presented at *IEICE USN Workshop*, July 2008. (in Japanese).

[69] E. Bonabeau, A. Sobkowski, G. Theraulaz, and J.-L. Deneubourg, "Adaptive task allocation inspired by a model of division of labor in social insects," in *Proceedings of Biocomputing and Emergent Computation*, pp. 36–45, 1997.

[70] M. Sasabe, Y. Taniguchi, N. Wakamiya, M. Murata, and H. Miyahara, "Proxy caching mechanisms with quality adjustment for video streaming services," *IEICE Transactions on Communications Special Issue on Content Delivery Networks*, vol. E86-B, pp. 1849–1858, June 2003.

[71] "Darwin Streaming Server." `http://developer.apple.com/darwin/`.

[72] "RealOne Player." `http://www.real.com/`.

[73] "QuickTime Player." `http://www.apple.com/quicktime/`.

[74] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata, "RTP payload format for MPEG-4 audio/visual streams," *Internet Request for Comments 3016*, Nov. 2000.

[75] M. Jain and C.Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," in *Proceedings of ACM SIGCOMM 2002*, pp. 295–308, Aug. 2002.

[76] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proceedings of ACM SIGCOMM 2003*, pp. 39–44, Oct. 2003.

[77] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: efficient available bandwidth estimation for network paths," in *Proceedings of the 4th Passive & Active Measurement Workshop (PAM 2003)*, Apr. 2003.

[78] D. Antoniades, M. Athanatos, A. Papadogiannakis, E. Markatos, and C. Dovrolis, "Available bandwidth measurement as simple as running wget," in *Proceedings of the 7th Passive & Active Measurement Workshop (PAM 2006)*, Mar. 2006.

[79] N. Yeadon, F. Gracía, D. Hutchinson, and D. Shepherd, "Filters: QoS support mechanisms for multipeer communications," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1245–1262, Sept. 1996.

[80] "NIST Net." `http://snad.ncsl.nist.gov/itg/nistnet/`.

[81] K. Fukuda, N. Wakamiya, M. Murata, and H. Miyahara, "QoS mapping between user's preference and bandwidth control for video transport," in *Proceedings of the 5th IEEE International Workshop on Quality of Service (IWQoS'97)*, pp. 291–302, May 1997.

[82] S. Lindsey, C. S. Raghavendra, and K. Sivalingam, "Data gathering in sensor networks using the energy*delay metric," in *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01)*, pp. 2001–2008, Apr. 2001.

[83] S. Lindsey and C. S. Raghavendra, "PEGASIS: Power-efficient gathering in sensor information systems," in *Proceedings of 2002 IEEE Aerospace Conference*, vol. 3, pp. 3–1125–3–1130, Mar. 2002.

[84] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the Hawaii International Conference on System Science*, pp. 3005–3014, Jan. 2000.

[85] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wirelss Communications*, vol. 1, pp. 660–670, Oct. 2002.

[86] K. Dasgupta, K. Kalpakis, and P. Namjoshi, "An efficient clustering-based heuristic for data gathering and aggregation in sensoer networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2003)*, vol. 3, pp. 16–20, Mar. 2003.

[87] S. D. Muruganathan, D. C. F. Ma, R. I. Bhasin, and A. O. Fapojuwo, "A centralized energy-efficient routing protocol for wireless sensor networks," *IEEE Radio Communications*, vol. 43, pp. 8–13, Mar. 2005.

[88] L. Gatani, G. L. Re, and M. Ortolani, "Robust and efficient data gathering for wireless sensor networks," in *Proceedings of the 39th Hawaii International Conference on System Sciences*, pp. 1530–1605, Jan. 2006.

[89] C. Intanagonwiwat, A. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, pp. 56–67, Aug. 2000.

[90] J. Heidemann, F. Silva, and D. Estrin, "Matching data dissemination algorithms to application requirements," in *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pp. 218–229, Nov. 2003.

[91] F. Silva, J. Heidemann, D. Estrin, and T. Tran, "Directed diffusion," *Technical Report ISI-TR-2004-586, USC/Information Science Institute*, Jan. 2004.

[92] M. B. H. Rhouma and H. Frigui, "Self-organization of pulse-coupled oscillators with application to culstering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 180–195, Feb. 2001.

[93] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pp. 95–107, Nov. 2004.

[94] "IEEE 802.15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs)," 2003.

[95] "CC2420." `http://www.chipcon.com/files/CC2420_Data_Sheet_1_4.pdf`.

[96] K. H. Low, W. K. Leow, and J. Marcelo H. Ang, "Task allocation via self-organizing swarm coalitions in distributed mobile sensor network," in *Proceedings of 19th National Conference on Artificial Intelligence (AAAI-04)*, pp. 28–33, July 2004.

[97] T. H. Labella and F. Dressler, "A bio-inspired architecture for division of labour in SANETs," in *Proceedings of the 1st International Conference on Bio-inspired Models of Network, Information and Computing Systems (BIONETICS 2006)*, Dec. 2006.

[98] "TinyOS." available at `http://www.tinyos.net/`.