A TRANSPORT PROTOCOL FOR REAL-TIME APPLICATIONS IN WIRELESS NETWORKED CONTROL SYSTEMS

By

LI GUI

BEng

A THESIS SUBMITTED TO

FACULTY OF SCIENCE AND TECHNOLOGY,

QUEENSLAND UNIVERSITY OF TECHNOLOGY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER BY RESEARCH

September 2010

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature

Date

To my parents

Keywords

Transport Protocol, Wireless Network, Networked Control Systems,
Real-Time Systems, Time Delay, Packet Loss,

Abstract

A Networked Control System (NCS) is a feedback-driven control system wherein the control loops are closed through a real-time network. Control and feedback signals in an NCS are exchanged among the system's components in the form of information packets via the network. Nowadays, wireless technologies such as IEEE802.11 are being introduced to modern NCSs as they offer better scalability, larger bandwidth and lower costs. However, this type of network is not designed for NCSs because it introduces a large amount of dropped data, and unpredictable and long transmission latencies due to the characteristics of wireless channels, which are not acceptable for real-time control systems. Real-time control is a class of time-critical application which requires lossless data transmission, small and deterministic delays and jitter. For a real-time control system, network-introduced problems may degrade the system's performance significantly or even cause system instability. It is therefore important to develop solutions to satisfy real-time requirements in terms of delays, jitter and data losses, and guarantee high levels of performance for time-critical communications in Wireless Networked Control Systems (WNCSs).

To improve or even guarantee real-time performance in wireless control systems, this thesis presents several network layout strategies and a new transport layer protocol. Firstly, real-time performances in regard to data transmission delays and reliability of IEEE 802.11b-based UDP/IP NCSs are evaluated through simulations. After analysis of the simulation results, some network layout strategies are presented to achieve relatively small and deterministic network-introduced latencies and reduce data loss rates. These are effective in providing better network performance without perfor-

mance degradation of other services. After the investigation into the layout strategies, the thesis presents a new transport protocol which is more efficient than UDP and TCP for guaranteeing reliable and time-critical communications in WNCSs.

From the networking perspective, introducing appropriate communication schemes, modifying existing network protocols and devising new protocols, have been the most effective and popular ways to improve or even guarantee real-time performance to a certain extent. Most previously proposed schemes and protocols were designed for real-time multimedia communication and they are not suitable for real-time control systems. Therefore, devising a new network protocol that is able to satisfy real-time requirements in WNCSs is the main objective of this research project.

The Conditional Retransmission Enabled Transport Protocol (CRETP) is a new network protocol presented in this thesis. Retransmitting unacknowledged data packets is effective in compensating for data losses. However, every data packet in real-time control systems has a deadline and data is assumed invalid or even harmful when its deadline expires. CRETP performs data retransmission only in the case that data is still valid, which guarantees data timeliness and saves memory and network resources. A trade-off between delivery reliability, transmission latency and network resources can be achieved by the conditional retransmission mechanism. Evaluation of protocol performance was conducted through extensive simulations. Comparative studies between CRETP, UDP and TCP were also performed. These results showed that CRETP significantly: 1). improved reliability of communication, 2). guaranteed validity of received data, 3). reduced transmission latency to an acceptable value, and 4). made delays relatively deterministic and predictable. Furthermore, CRETP achieved the best overall performance in comparative studies which makes it the most suitable transport protocol among the three for real-time communications in a WNCS.

Acknowledgements

First and foremost, I owe my deepest gratitude to my supervisor, Associate Prof. Yu-Chu (Glen) Tian, for his invaluable expertise, wisdom, patience, encouragement throughout the course of this research project. Without his helpful guidance, this work would not have been accomplished. Furthermore, his many refinements to this thesis have been much appreciated.

Many thanks also go to my associate supervisors, Prof. Colin J. Fidge and Dr Hasmukh Morarji for their generous support and comments on my working during this candidature. I am also indebted to my fellow graduate students, Gus Tian and Mark Miao for their support and friendship.

Special thanks to Discipline of Computer Science, Faculty of Science and Technology, Queensland University of Technology (QUT) for providing the support, workstations and software needed to produce and complete my thesis.

Finally, I wish to extend my appreciation to my family for their love and support during the completion of the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis, entitled "A Transport Protocol for Real-time Applications in Wireless Networked Control Systems", focuses on improving real-time performance of control systems that adopt wireless technologies such as IEEE802.11 standards. The overall objective of the research project described in this thesis is to meet real-time requirements by providing reliable data transmission and guarantee data timeliness for real-time control applications. To achieve this objective, this thesis developed a network protocol based on modification of User Datagram Protocol (UDP) and several strategies for device deployment in WNCSs. This chapter introduces the research background, the motivation and the aims of the project. It also highlights the research significance and main contributions of this thesis.

1.1 Research Background

A networked control system (NCS) consists of sensors, actuators and controllers that communicate over a network to exchange data for monitoring, controlling and synchronizing industrial processes. This class of control system is important in manufacturing and industrial process control. It began with the traditional centralized

point-to-point control system. A point-to-point network has the characteristics of small transmission delay and deterministic variation of latency (jitter) which suits real-time control systems very well. However, they do not scale well and require much effort in maintenance. Later on, the common-bus network architecture was introduced in NCSs. In this type of architecture, a wired fieldbus achieves higher flexibility, and less installation and maintenance cost than a point-to-point network and still provides deterministic and timely communications.

With the rapid developments in communication technologies, some data technologies such as Ethernet and the wireless IEEE 802.11 standard have been used in modern NCSs as they offer better scalability, larger bandwidth and lower cost. When compared with a wired NCS, a wireless networked control system provides fast deployment, and low implementation and maintenance costs which make it very popular in industrial process control [Nilsson, 1998]. However, there are significant differences between a control network and a data network. In particular, wireless networks distinguish themselves by many additional characteristics such as interference, channel variations and channel outage. These features introduce longer transmission delays as well as higher rates of packet loss. This is not favourable for real-time control systems. Therefore, a real-time and reliability requirements are the key issues that must be resolved in a wireless NCS.

Generally speaking, there are two types of real-time communication that are required in real-time control systems [Stoina, 2008]. For the first type, the transmission time is important but not crucial, while the variation of delay (jitter) is critical and has to be within an acceptable bound to guarantee acceptable control performance. In the second type of real-time communication, a control system must ensure that the end-to-end delay of a packet transmission stays within the pre-defined processing time cycle, or period, as the information carried by the packet has a limited validity. In addition, solutions for data dropouts are needed to ensure the receiver always gets the control signals sent. Therefore, making transmission delays deterministic and predictable, reducing transmission latency to an acceptable value, and avoiding or compensating

for packet dropouts are essential for performance guarantees and reliability of time-critical applications in wireless NCSs.

Recently, many methodologies have been developed to provide a guaranteed quality of service (QoS) in real-time networked control systems. Each of them focuses on different aspects of QoS. However, it is still a new research area and a lot more work needs to be done.

1.2 Statement Gap and Motivation

When wireless networks such as 802.11 networks are applied as replacements for hard-wired ones in an NCS by many enterprises nowadays, a larger amount of dropped data, and unpredictable and longer transmission latencies are introduced in the communication network due to the characteristics of wireless channels. These drawbacks may degrade the system's performance significantly or even cause system instability if the applications in the NCS are real-time control applications. Real-time control is a class of time-critical application which suffers more than others from unpredictable or long transmission delays, and packet losses. It is therefore important to develop solutions to satisfy real-time requirements and guarantee high levels of performance for time-critical applications.

It is known that soft real-time applications are delay sensitive but can tolerate isolated errors and losses and error/loss bursts. Usually, soft real-time focuses on mean times and tries to reduce deadline miss ratios [Watteyne and Auge-Blum, 2005]. Therefore, the combination of low overheads and discarding packets rather than retransmitting them allows more frequent transmission of small packets, making UDP a preferred choice for soft real-time applications in NCSs. It is not surprising that most real-time applications use UDP as their transport layer protocol instead of TCP since this kind of application cannot afford the latency caused by the retransmission function. However, UDP is a connectionless protocol which cannot provide a reliable packet delivery service. No acknowledgement is sent back by destination nodes and

the source nodes do not know whether packets are successfully received or not. This could lead to excessive packet losses and result in a large dropout ratio which is not favourable for soft real-time applications.

Dealing with packet dropouts, and reducing transmission delays and jitter are the most important aspects in real-time requirements that will be focused on in this research. Introducing appropriate communication schemes, modifying existing network protocols and devising new protocols, have been effective and popular ways to improve or even guarantee real-time performance to a certain extent.

Network protocols working on different network layers are developed to improve performance of real-time communication. On the application layer, the real-time transport protocol (RTP) is well known and in widespread use in communication and entertainment systems. It has the ability to detect out-of-order arrival in data stream and provides a strategy for jitter compensation in the application layer. Since RTP is designed for real-time multimedia communications, it defines its standardized message format for delivering video and audio over internet.

Some transport layer protocols based on modification of UDP, such as UDP-Lite and CUDP, have been developed to compensate for the unreliable transmission of UDP. UDP-Lite – a lightweight version of UDP with increased flexibility in the form of a partial checksum [Larzon et al., 1999]– provides the flexibility of a partial checksum and does not drop corrupted frames, thus minimizing the drop burst length and improving throughput [Larzon et al., 1999]. Khayam et al [2003a] introduced MAC-Lite to abandon retransmissions and pass partially corrupted frames to higher layers. It functions together with UDP-Lite to achieve improvements in media quality. However, UDP-Lite has two major drawbacks. One is backward incompatibility which makes traditional UDP applications useless if UDP-Lite is employed in a system. The other one is that there are no application layer protocols that can handle the corrupted packets passed from UDP-Lite [Lam and Liew, 2004].

CUDP was proposed by Zheng and Boyce to use information of channel errors obtained from the physical and link layers to help error recovery at the packet level [Zheng

and Boyce, 2001]. It outperforms the UDP and the UDP-Lite protocols. Unfortunately, the advantage of CUDP reduces as the congestion packet loss rate grows.

New MAC layer protocols have been proposed for real-time applications in ad hoc wireless networks with focus on achieving reductions in transmission latencies, providing timely delivery guarantees, and scheduling channel accesses for real-time sessions, respectively [Baldwin et al., 1999; Cunningham and Cahill, 2002; Pal et al., 2002a; Zhou et al., 2003].

A Burst-oriented Transfer with Time-bounded Retransmission (BTTR) scheme working on the link level was proposed by Wu [2001] which employs a large transmission window for sending and receiving a burst of time-critical data. It can be helpful to meet the delay and throughput requirements; however, obvious degradation on the packet loss rate occurs.

There are not any very effective methods that can reduce end-to-end packet loss rates in wireless NCSs. Also it is worth considering if there is any strategy that would satisfy the delay requirements without degradation of performance from other aspects in network communication. These issues motivated our proposed research project with regard to resolving these two problems:

1. *How to achieve small transmission latency and jitter of data communication by properly deploying devices in the WNCS;*

2. *How to develop a new transport layer protocol which would improve the reliability of transmission for real-time applications by compensating for packet losses while keep data timeliness in the WNCS.*

1.3 Aims of the Research

When data networks are introduced into NCSs, many challenges turn up in the design of real-time NCSs from both control and networking perspectives. The objective of our research is to provide strategies that improve the network performance of real-time wireless NCSs, implying that control related issues such as the dynamics of the

controller and processes to be controlled, are beyond the scope of our work. Several aims are summarized below:

1. *We aim to present strategies about how the network layout, transmission interval of the sensor and other network characteristics affect the transmission latency and jitter. A network deployed according to the proposed rules, should achieve the minimum transmission delay and jitter;*

2. *We aim to design a transport protocol that has provision for packet retransmission. This new protocol must be capable of anglicizing current end-to-end round trip latency and providing retransmissions within the present time limit. By doing so, it will decrease the data loss ratio without violating data timeliness;*

3. *We then need to evaluate the performance of the proposed protocol by comparing the protocol's performance with performances of other transport layer protocols used now.*

The aims of our research are proposed based on network requirements specified when designing a real-time wireless NCS. Network performance can be improved with the achievement of those aims. However, the overall performance of an NCS is not evaluated since the control performance determined by control methodologies is not analysed in this research.

1.4 Significance of the Study

In real-time networked control systems, both network Quality-of-Service (QoS) and controller's Quality-of-Control (QoC) are expected to be guaranteed. On the network side, throughput is not the most important aspect of network QoS. Instead, transmission latency and reliability become more critical, especially when the control algorithm in the NCS is delay sensitive. Real-time control is a class of time-critical application which suffers more than others from unpredictable transmission delays and unreliable data transmission. Reducing transmission latency to an acceptable value, making

delays deterministic and providing reliable communications are major focuses in this research.

From the perspective of transmission latency, the challenges are caused by variable and comparatively long times of data transfer between devices induced by introducing a wireless network as the communication technology. In an NCS, a control packet in a real-time application becomes worthless or even dangerous if it is not received within the pre-defined deadline. Furthermore, the variation of delay (jitter) has to be within an acceptable bound to guarantee stable control performance. From the aspect of NCS deployment, proper design of the network layout can achieve better network performance without performance degradation of other services. Several strategies of device arrangement in WNCSs presented in our study can be effective in reducing the time of data transmission, jitter and data dropout rate. When most data packets can be successfully interchanged between devices within a small and certain time limit, the performance of a real-time application can be improved.

From the perspective of transmission reliability, the challenge is caused by the User Datagram Protocol used by most real-time applications in NCSs. UDP ensures a fast transmission so that a small latency can be achieved. However, it is connectionless and cannot provide reliable delivery. If many data packets are dropped during transmission and no compensating methods are there to make up for the packets' loss, the system's performance will significantly degrade or even worse, the system becomes unstable. In our research, a new transport layer protocol is designed based on UDP but providing a reliable delivery service. This protocol employs the concept of retransmission and acknowledgement while keeping data timeliness in mind. Retransmission is one of the most effective ways of compensating for lost packets. However, retransmission causes unpredictable latency of delivery and uses extra resources such as memory and network bandwidth. Therefore, the new protocol enables retransmission conditionally so that a trade-off between delivery reliability, transmission latency and network resources can be achieved. Evaluation of this protocol is done by a large number of simulations. The results are evidence that this protocol significantly improves the reliability of

communication especially when the condition of the wireless channel is not good.

The proposed network deployment rules and the new protocol can be employed together in a wireless NCS to improve system performance in terms of delay, jitter and transmission reliability as they are designed from totally different aspects.

1.5 Contribution of the Thesis

As the aims of the research have been fully accomplished, the main contributions of this thesis are listed as follows:

1. *Strategies about how to deploy the network, taking into account the number of sensors and sensors' data rates, to achieve a better performance in regard to transmission latency and jitter are developed;*

2. *A transport layer protocol for real-time communication in a wireless NCS is developed. It enables conditional retransmissions so that the reliability of communication can be significantly improved without violating data timeliness.*

1.6 Thesis Organization

The rest of this thesis is structured as follows. Chapter 2 presents a detailed survey of the existing literature and background information in related research areas. It identifies the focuses and weaknesses of these previous papers. Chapter 3 discusses the relationships between network layout, data rate and transmission latency based on simulations. Several strategies for network deployment are proposed which can help reduce transmission delays, jitters and data losses. As one of the main contributions of this thesis, Chapter 4 presents the design of a new transport layer protocol featuring conditional retransmission. Logical algorithms and protocol mechanisms are elaborated. Evaluation of the new protocol through extensive simulation is done in Chapter 5 which also includes comparative studies between the proposed protocol and other transport protocols used now. Both advantages and disadvantages of this

protocol are analysed. Chapter 6 concludes this thesis and discusses the direction for further research on this topic. The main part of the protocol implementation code is given in an appendix.

1.7 Related Publications

- L. Gui, Y.-C. Tian and C. Fidge. Performance evaluation of IEEE 802.11b wireless networks for networked control systems. In *Proceedings of the 2007 International Conference on Embedded Systems and Applications (ESA07)*, pages 121-126, Las Vegas, Nevada, USA, 25-28 June 2007.

- A journal paper "QoC Elastic Scheduling for Real-Time Process Control Systems" has been submitted to Real-Time Systems.

Chapter 2

Previous and Related Work

Networked control systems are a kind of computer-controlled system. Computers began to control processes directly when the direct digital control (DDC) system was developed [Zhang, 2001]. DDC systems use point-to-point wiring which is expensive and difficult to maintain and scale. Later on, the distributed control system (DCS) came out for fast deployment of computer technology and distribution of computation power. When computers were able to control entire production plants, and network interfaces were enabled with sensors and actuators, networked control systems (NCSs) were formed carrying real-time mixed traffic. A Networked Control System is a feedback-driven control system wherein the control loops are closed through a real-time network, as illustrated in Figure 2.1. The defining feature of an NCS is that control and feedback signals are exchanged among the system's components in the form of information packets through a network. The functionality of a typical NCS is established by the use of four basic elements: sensors that acquire information, controllers that provide decision and commands, actuators that perform the control commands and a communication network which is used to enable exchange of information [Ye, 2000].

Communication networks affect the dynamic behaviour of control systems in an NCS [Baillieul and Antsaklis, 2007]. Traditionally, the communication network of an

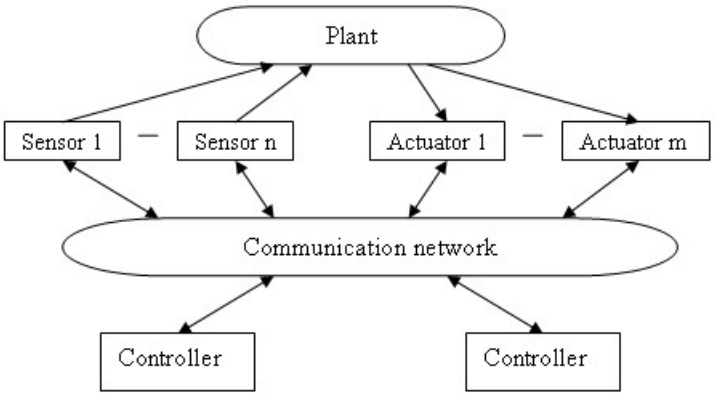

Figure 2.1: Generic setup of an NCS

NCS is a local area network (LAN) or a wide area network (WAN). After wireless technology came out, wireless local area networks (WLANs) were proposed as an extension to or as an alternative for LANs in situations where it is difficult or impossible to install wireline connections [Jiang, 1998]. Compared with conventional point-to-point control systems, modern NCSs have many advantages such as improvements to system scalability, reduction of weight, space and wiring requirements. However, signal delays, distortion and loss caused by finite bandwidth constraints and competition between multiple system components are drawbacks. More challenges exist when designing and implementing a WLAN in an NCS such as interference caused by collisions and multi-path fading, low bandwidth, time-varying throughput, limited power and mixed traffic in communication channels [Ye et al., 2001].

Uncertain transmission delays and packet losses are unavoidable when random access networks such as Ethernet and WLANs are applied in NCS. Real-time applications require the data transmitted between devices in NCSs to be accurate, timely, and lossless. The worst case scenario in a real-time NCS is that the transmission time of a control signal is large and unbounded, and even that packet losses occur. Therefore, it is necessary and important to understand and improve the real-time performance of network communications in NCS. The following sections elaborate two major parameters of network performance for real-time applications: transmission delay and

reliability, and we then focus on research into delay/jitter reducing methods and reliability improvement schemes for real-time applications in wireless communication respectively.

2.1 Time Delay in Networked Control Systems

For a real-time control system, two essential characteristics are that the control system must produce correct computational results, called logical or functional correctness, and that these computations have to be done within a predefined time cycle, called timing correctness [Li and Yao, 2003]. Sometimes, timing correctness is more important and functional correctness may be sacrificed for it. Timing problems is divided into control delays, jitter, and transient errors [Wittenmark et al., 1995]. From the point of view of a network, jitter represents latency variation and occurs when delays are not constant.

Generally speaking, a control delay contains three parts, illustrated in Figure 2.2. They are the communication delay between the sensor and the controller (T_{sc}), the communication delay between the controller and the actuator (T_{ca}), and the computation time in the controller (T_c). In this research, we focus on communication delays between devices as the computation delay is not affected by networks in NCSs. Communication latency of data exchanges between sensors, actuators and controllers over the network is defined as network-induced delay in an NCS [Cervin et al., 2003].

In Figure 2.3, a message transmission delay in NCSs is defined more clearly when divided into several different components. Hristu-Varsakelis and Levine [2005]pointed out that the packet transmission delay (T) is composed of the pre-processing time at source devices (T_0); the packet's waiting time in the queue at the sender's buffer (T_1), the post-processing time at destination devices (T_3), and the packet transmission time in the network medium (T_2). The relationship between these delays is as shown below:

$$T = T_0 + T_1 + T_2 + T_3$$

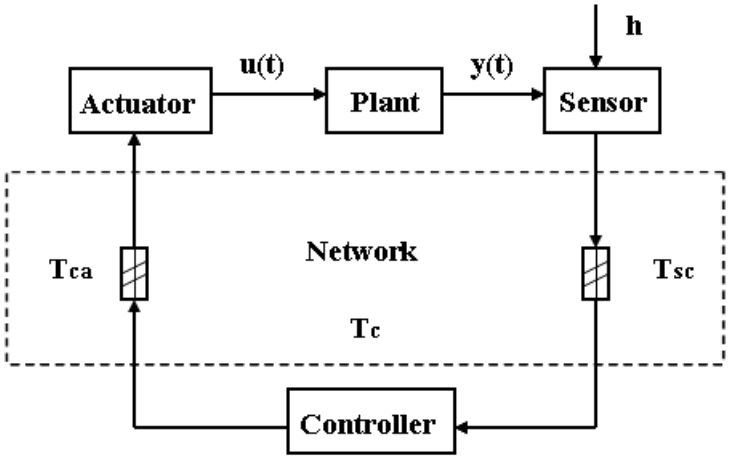

Figure 2.2: Delays in an NCS

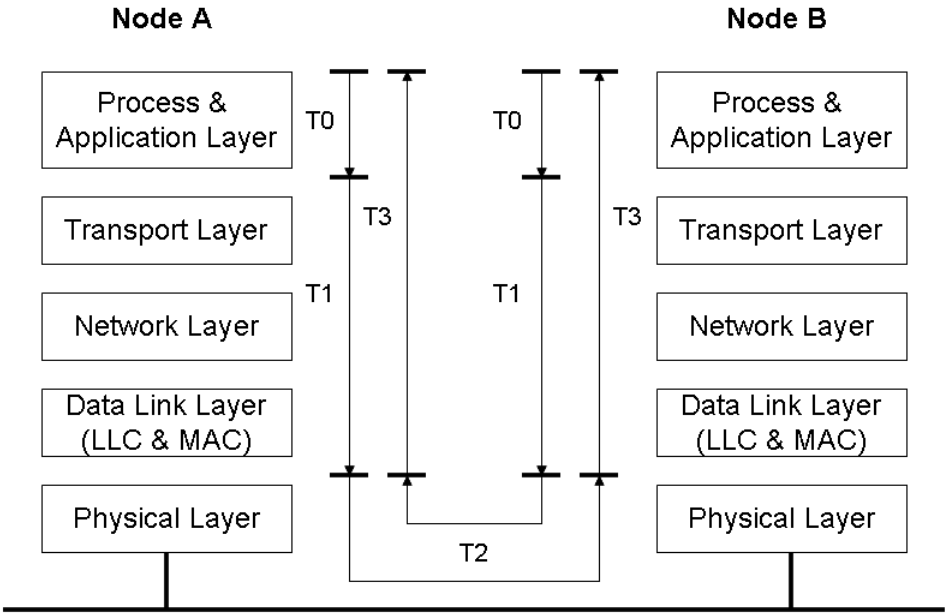

Figure 2.3: Components of an end-to-end delay

Among all these timing components, the pre-processing time (T_0) includes the time to acquire data from the external environment and the data encoding time. Both T_0 and T_3 depend on the software and hardware characteristics of sending and receiving devices which have little to do with networking [Wen et al., 2007]. T_2 includes the

propagation time between two devices and the frame transmission time. The propagation time is negligible as it is usually a very tiny part of end-to-end delays. The frame transmission time can be calculated given the size of the data, overhead and padding and the data rate. Therefore, T_2 is the most deterministic part. There are also two parts of a waiting time (T_1). One is the time a packet waits until it gets to the top of the buffer. The second part is the time during which a packet is blocked from transmission by other data on the network. It is affected by the amount of data being sent at the source node, the network protocols and the network traffic load [Gupta et al., 2007]. When devices in the NCS are connected with random access networks such as Ethernet, or the IEEE standard 802.11, the transmission time of data between devices is random and is generally considered non-deterministic. The significant parts of uncertain network delays are the waiting time delays (T_1) introduced by queuing and collisions on the network [Tipsuwan and Chow, 2003].

Previous analysis of network transmission delays did not take into account the type of network and the network protocols used in NCSs. Our research focuses on real-time data transmission in a WNCS with IEEE 802.11 technology. Therefore, we elaborate the waiting time (T_1) on Mac layer and the transport layer with the protocols used in the 802.11 standard.

The 802.11 standard specifies a medium access control (MAC) layer which coordinates access to a shared wireless channel. Two forms of medium access functions are defined and utilized. They are the point coordination function (PCF) and the distributed coordination function (DCF). PCF is regarded as unsuitable for real-time data due to the overhead in IEEE 802.11's centralized access scheme [Hsu and Chen, 2006]. DCF is mandatory and based on the CSMA/CA (carrier sense multiple access with collision avoidance) protocol. With DCF, stations in an 802.11 network contend for medium access and send packets when the channel is idle. A random back-off timer is used by the station to detect the state of the medium before sending data. When a busy channel is detected, the station must wait a random period of time and attempt to access the medium again. This random back-off time can be affected by the number

of communication devices in the network, the transmission rates of active devices, the channel's condition, etc.

The Transport layer provides delivery of data to the appropriate application process on the destination device, error checking mechanisms and data flow controls. TCP and UDP are the most well known transport layer protocols and are used in both wired and wireless networks. Retransmission is the key function of TCP which makes it a reliable transmission protocol. TCP also rearranges out-of-order packets, and even helps minimize network congestion through flow control with variable-size sliding windows. However, all these functions incur relatively long transmission delays and heavy packet loads. Therefore, TCP is optimized for accurate delivery rather than timely delivery. As opposed to TCP's complexity, UDP is light-weight with minimal overhead and fast transmission as it does not implement reliable/ordered delivery or flow control. Real-time applications apply UDP on the transport layer for its fast transmission but this leaves reliability as the problem to be solved.

Delays in a control loop have harmful effects on a control system, as do the network delays in an NCS. Liu and Goldsmith pointed out that control performance degrades due to distributed medium access control in addition to the performance degradation caused by random delays and frame losses [Liu and Goldsmith, 2004]. Another harmful effect of random delays in an NCS is system destabilization. System performance degradation by delays in-the-loop and how the delays can reduce the stability region of the system are illustrated by Wittenmark et al [1995]. More analysis work of stability for NCSs with random network delays is presented by Cloostermann et al [2008,2009]. Therefore, in order to prevent control performance degradation and prevent destabilization of the control system, considerable research is being done to minimize or compensate for random delays and jitter.

2.2 Packet Loss in Networked Control Systems

The term "packet loss" defines the failure of one or more transmitted packets to arrive at their destination. A data packet may be dropped at any point across the network link or could be discarded by the destination if it fails the checksum. Packet loss is one of the main error types encountered in all types of digital communications especially in media that are prone to transmission errors like a wireless medium [Hirano and Murase, 2008]. The average packet loss rate for a network connection gives an overall sense of the quality of the connection.

There are numerous reasons why packet loss occurs. Some key reasons are as follows:

- Network congestion. For example, a certain period of network congestion makes the router buffers saturate which at some point leads to a situation where there is no room to store more packets in the buffers, so newly arriving packets get dropped;

- Errors in physical network links (which is relatively common in wireless networks). Wireless networks introduce longer delays and higher rates of packet loss when compared with wired networks due to their variable channel quality over time and space. The variations are caused by either motion of the wireless device, or changes in the surrounding physical environment, etc [Shakkottai et al., 2003]. Signal degradation can easily happen over the network medium due to a fading channel and a weak signal leads to packet dropouts;

- Router configuration. For example, some routers employ a mechanism called random early detection (RED) in order to implement rate control by dropping data packets. Such dropouts are a way of communicating back to the senders the need to scale back the offered load on the network;

- Faulty networking hardware, normal routing routine failures or software corruption are common causes of packet loss.

Often more than one of these factors is involved in a packet dropout. In NCSs, transmission delays can play a role in introducing dropouts. For example, long time-delays may result in packet reordering, which amounts to a packet loss if the receiver rejects the arrival of out-of-date packets [Hespanha et al., 2007]. From a networking aspect, packet losses can be categorized as random losses or burst losses due to congestion. According to the type, or pattern, of packet losses, there are at least two kinds of loss that have been studied intensively: single loss and burst loss. Wu [2006] found that the probability of losing a single packet is the largest.

The data loss ratio is a criterion of transmission reliability. Unreliable transmission of data results in negative effects in different ways. In text and data, a packet loss produces errors. In audio/video communications, it creates short-term gaps. When packet loss occurs in a real-time control system, a sequence of harmful effects is introduced. There is no doubt that the loss of a control related packet may lead to the complete absence of a control signal which results in increasing data and control delays. Jitter occurs if the time-delay varies. Increased delay and jitter degrade control performance and may cause system instability. Packet loss is also the cause of transient faults in an NCS. Wittenmark et al [1995] indicated that a temporary blackout of the control system can be caused by a transient fault. This is a very serious system error as the control system will behave in an unpredictable way for a period of time, such as no action at all or erroneous action.

In a wireless network, packet losses are more likely to happen due to the comparatively unstable state of the medium when compared to a wired network. For example, wireless medium contention and the hidden terminal problem at the MAC layer may lead to link-layer packet dropping [Tsigkas and Pavudou, 2008]. In an NCS, UDP is employed instead of TCP as it meets the requirements of delay-sensitive real-time applications. However, UDP does not provide reliable transmission and does nothing when packet loss happens. Therefore, packet loss is a serious problem to be solved in wireless NCSs.

2.3 Compensation for Transmission Delay

As the devices in an NCS are not point-to-point linked nowadays, the transmission time is random and generally considered non-deterministic. When wireless technology is introduced to network control systems, more effort must be put into real-time guarantees for delay sensitive applications because wireless networks introduce longer and random delays as well as higher rates of packet loss when compared with wired ones. One challenge for the network in an NCS with real-time applications is to provide deterministic and small delays, and another challenge is to provide reliable transmission. Quality-of-Service (QoS) will not be guaranteed if these two challenges cannot be fulfilled.

Yang and Wang [2005] identified two types of solutions to the reduction of network-introduced delay and jitter. The first one is optimizing the Quality-of-Control of the control system with methodologies from control theory that act against the harmful impaction of transmission latency. The other solution is to improve the Quality-of-Service of the network by employing a reliable communication protocol and adopting devices with high performance.

From the aspect of Quality-of-Control optimization, much NCS research has been conducted on controller design to provide sufficient stability margins in the presence of the network-induced effects in a unified model [Cervin et al., 2006; Dermanovic et al., 2004]. This type of methodology can be effective in compensating for transmission delays and packet losses only if the plant dynamics are well understood and accurate models of the plant and communication network are built. However, the complexity of the controller design and communication network modelling increases with the increase in control system complexity. When random access network technologies such as Ethernet and IEEE 802.11 replace the traditional communication method in an NCS, it is often difficult to model network delays and data dropouts with formulations based on the characteristics and the probability theories of network devices [Dermanovic et al., 2004].

In an NCS, controller design and network modelling to compensate for transmis-

sion delays need an understanding of delay behaviours in advance. Previous work on network-induced delay analysis was usually done with the assumption that the delay is constant, or time-varying but periodic, or random but bounded, or even known by the controller.

Compensation for delays is simplified if the delay can be a fixed constant. Early in the 1990s, a buffering procedure was introduced by Luck and Ray [1990]. The network-induced delay was regarded as time-invariant by employing buffers longer than the worst case delay. In this case, it is not difficult to maintain the system's stability as the system can be considered time-invariant. This idea was extended in further research [Luck and Ray, 1994] and a delay-compensation algorithm using a two-steps observer/predictor was formulated to handle the time-varying delays. Some early work on compensation for time-varying delays in an NCS was done with the assumption that delays are periodic as they are introduced by the communication network due to deterministic scheduling and queuing [Zhang et al., 2001]. However, neither constant nor periodic delays are a good model for an NCS with random access networks, where random delays are generated.

Both Ethernet and WLAN are random access networks and are being widely used as communication networks in NCSs. Various control methodologies have been discussed to provide timely transmission of control signals in an Ethernet based NCS such as the queuing methodology [Tian et al., 2006]. As mentioned at the beginning of this section, employing proper network communication protocols is another way to compensate for delay and jitter. Protocols which enable hard real-time communication on Ethernet have also been proposed [Loeser and Haertig, 2004; Ouni and Kamoun, 2002; Yiming and Eisaka, 2005].

If the communication network in an NCS is a wireless one with the IEEE 802.11 standard, it is still a challenging job to provide time constrained delivery of data for real-time applications since a reasonable upper bound for transmission times cannot be found. To provide timely transmission in a WLAN, modifications to the IEEE 802.11 protocols are necessary.

Some new MAC layer protocols have been developed which reduce the missed deadline ratio of real-time data in ad hoc wireless networks. In an ad hoc network with the IEEE 802.11 standard, packet collisions and the transmission of packets that have already missed their deadlines are two major factors that impact the missed deadline ratio [Baldwin et al., 1999]. Sending an out-of-date packet wastes channel capacity and could lead to failures of other packets meeting their deadlines. Packet collisions can be avoided by selecting or deferring a proper back-off value (BV). Baldwin and his colleagues proposed a modified IEEE 802.11 protocol, RT-MAC, which uses additional information to help a real-time WLAN meet packet deadlines [Baldwin et al., 1999]. These new pieces of information are: a transmission deadline of a packet and the transmitting station's next BV. The value of a packet's deadline is examined three times during its lifetime. A packet is first checked when it is in preparation for transmission and it will be examined again after the back-off time expires. If the packet deadline has not been exceeded, it is ready for transmission. The last examination happens when this packet is not successfully transmitted (no acknowledgement is received). Once the deadline is exceeded, the packet is discarded otherwise it will be retransmitted. This transmission control algorithm for RT-MAC theoretically guarantees the timeliness of a successfully received packet.

The enhanced collision avoidance algorithm (ECA) of RT-MAC reduces the probability of packet collisions and decreases the waiting time of packet transmissions in the MAC layer. In the ECA algorithm, the contention window (CW) is no longer a static value but eight times the number of stations in the network (the value of 8 is based on the CW equation used by Bianchi et al [1996]. This comparatively large CW causes less probability of two or more nodes selecting the same BV. Each packet that is to be sent will have a BV in its header chosen by the transmitting station. Therefore, other stations who hear the transmission will know that this BV is used. A station must change to a new BV if its current BV is the same as the received one. Through simulation, the missed deadline ratio, collision ratio and mean delay of successfully received packets decreased dramatically when the IEEE 802.11 MAC layer protocol

was replaced by the RT-MAC. RT-MAC prevents different stations from choosing the same BV to avoid packet collisions. However, it does not have any further control like a smaller value of the back-off time should be given to a more urgent packet.

Pal et al [2002b] introduced a prioritized collision avoidance mechanism and a deterministic collision resolution algorithm in the elimination by sieving protocol (ES-DCF) that supports hard real-time data traffic in ad-hoc wireless networks. This novel MAC layer protocol got its name in regard to the operation of a dynamic distributed sieve-like mechanism in the collision avoidance phase of channel access.

A variable called the graded channel-free-wait-time is used in ES-DCF which is suggested by Deng and Change [1999]. Each real-time packet has a graded channel-free-wait-time that is used by the station to decide when to send its RTS. The closer the deadline of the packet, the smaller is its graded channel-free-wait-time. Finer sub-grades can be created if two or more packets fall into the same grade. The value of the graded channel-free-wait-time decreases as the packet ages. By deploying this graded variable, ES-DCF allows packets with higher priority access to the channel earlier than those packets whose priority is lower. Collisions may still happen if two or more packets get the same grade or sub-grade. In this situation, ES-DCF uses the method of transmitting "black-burst" of signals. The length of the "black-burst" equals the ID number of the real-time node and is unique. When a collision happens, a node with a smaller ID number must defer its transmission attempt if it hears a black-burst of longer duration. It is recommended that real-time nodes with higher data-generation rates have greater node IDs. This collision resolution method is deterministic as the ID number of each real-time station decides its transmission priority. A more complicated window-splitting discipline proposed by Markowski and Sethi eat al [1998] can be used in collision resolution.

As ES-DCF is designed to improve the performance of hard real-time traffic, Pal et al extended their study by proposing the deadline bursting (DB-DCF) protocol which works specifically for soft real-time traffic. The DB-DCF [Pal et al., 2002a] enables a black-burst contention phase at the beginning of each channel access attempt. A node

has to send a black-burst of length before its real-time packet transmission. The length of the black-burst is affected by the deadline data packet. The closer the deadline, the larger is the black-burst's length. The node listens to the channel after its sending of a black-burst. If a longer-duration black-burst is heard, it defers data transmission, recalculates the burst length and starts a fresh black-burst contention cycle. If two or more nodes have sent the same value of black-burst length, a collision occurs. In such cases, a collision resolution phase is initiated. The collision resolution mechanism of DB-DCF is the same as that of ES-DCF with the only exception that all nodes with packets that experience collisions use the smallest channel-free-wait-time.

The specific distribution of channel-free-wait-time for each real-time packet makes ES-DCF good at improving performance of hard real-time data traffic in terms of average transmission latency and throughput, while DB-DCF focuses on timely delivery of soft real-time packets.

Besides these new MAC layer protocols, researchers also attempted to propose protocols on other layers in order to satisfy real-time applications in wireless networks. A link layer protocol that meets timing requirements of real-time data transmitted through high-speed link with non-negligible propagation delay was developed by Wu [2001]. This Burst-oriented Transfer with Time-bounded Retransmission (BTTR) scheme works on a burst-by-burst basis so that a large transmission window is employed for sending/receiving a burst of data. It adopted a Negative ACK instead of the traditional ACK. A transmission is recognized as successful if the negative ACK is not received within a predefined period. This scheme also limits the number of retransmission trials to only one for each burst of data. Both the negative ACK and limited retransmission are helpful to reduce transfer latency. However, on the other hand, transmission reliability is affected. The decrease in delays is achieved by an increase of the packet loss rate. Therefore, this burst-oriented transmission protocol is only suitable for soft real-time applications such as video and voice.

QoS for a real-time application has many aspects. Besides transfer latency and jitter, the reliability of transmission is also a key element. Some new methods fo-

cusing on improving reliability of real-time data transfer may result in degradation of network performance in term of timing. Therefore, schemes are needed to optimize the performance of these new methods in respect to transmission delays. Wang and Zhen [2010] developed a reliable dynamic buffer UDP scheme (RDBUDP) to improve the performance of Reliable UDP in the aspect of transmission delays and network throughput. Reliable UDP is a UDP based protocol designed to guarantee the reliability of UDP and is elaborated in the next section. The principle of RDBUDP is that the sender identifies the buffer size that the receiver should reserve to store the potentially disordered data before it receives the packets. When the buffer is full, the receiver should rearrange the disordered packets and submit them to the upper layer before it can start receiving new data packets. Unfortunately, Wang and Zhen [2010] did not present any algorithm showing how the buffer size should be defined every time. Lu [2001] gave an estimation of the buffer size using mathematical formula. However, the formula is too complex and the estimation is not accurate enough. Some other mechanisms designed to reduce the delay introduced by reliable transmission schemes for real-time applications are presented by Afonso and Neves [2004; 2005].

2.4 Techniques for Packet Loss Compensation

Guaranteeing of transmission reliability is another challenge in providing QoS for delay sensitive applications in a wireless NCS. However, network congestion and wireless channel failure yields tremendous data dropouts. As real-time applications can only afford a small number of packet losses, the performance of the system will be improved if the data loss ratio can be reduced. Some techniques for avoiding packet dropouts or taking appropriate action when a packet loss happens are proposed in recent research.

It has long been understood that a communication protocol is an effective way of maintaining and improving required performance of the whole network and individual applications. At the transport layer, there is no doubt that TCP is the most famous

protocol for reliable data communications. The best explanation of TCP is documented in the RFC 793 standard. It is a connection-oriented, end-to-end reliable protocol designed over an unreliable network. One of TCP's key functions is retransmission. A positive ACK is expected from the receiving TCP for the sender's successful data transfer. If the ACK is not received within a timeout period, the data are retransmitted. Lost or corrupted packets could be compensated for by retransmission. TCP also has the responsibility to ensure the correct order of received packets. However, the overhead of guaranteeing reliable and accurate data transfer reduces the network's throughput. Furthermore, TCP increases the average transmission delay and worsens the jitter [Liu et al., 2002]. Thus, TCP is not suitable for applications where delay is the first concern.

The UDP protocol is applied to delay sensitive applications instead of TCP for its fast transmission and light weight. To overcome the unreliability of UDP, some new protocols based on UDP were developed with schemes to reduce the data dropout ratio. UDP-Lite is a novel version of UDP. It was proposed as some researchers found that a number of "lost" packets had actually reached the in destination, but were discarded by the UDP protocol stack as they failed the checksum [Larzon et al., 1999]. Using UDP-Lite can reduce the number of discarded packets which makes the dropout ratio decrease. The reason why packet discards do not happen very often for UDP-Lite is that UDP-Lite allows for partial checksums. It only verifies the error-sensitive portion of a UDP datagram. This portion is identified by the coverage field in the UDP-Lite header. If the error-sensitive part of the datagram does not fail the checksum, the whole datagram will be delivered to the applications. In this case, errors in the insensitive portion of the datagram are ignored and the packets that are delivered to the applications may have been partially corrupted. The applications may or may not fix the errors. As UDP-Lite is designed for multimedia applications, a packet with unfixable errors only causes a glitch and it not very harmful. Therefore, UDP-Lite is able to improve the overall performance and quality of many UDP based soft real-time applications [Singh et al., 2001].

However, there are obvious drawbacks of UDP-Lite that limit its effective use. Larzon et al [2004] recognized the backward incompatibility problem of UDP-Lite. The coverage field in a UDP-Lite header is essential for this protocol and it is changed from the "UDP Length" field in a traditional UDP header. This replacement causes a UDP-Lite header to be incorrectly interpreted by a traditional UDP application.

Also, without the cross-layer interaction between TCP/IP stacks and lower layers such as the link layer, lower layers are not aware of the new checksum calculation of UDP-Lite. Modern link layers with a strong cyclic redundancy check (CRC) will discard damaged packets before these packets can be delivered to the transport layer. Khayam et al [2003a] evaluated the suitability of a cross-layer design for UDP-Lite. The authors suggested the combined use of MAC-Lite and UDP-Lite. The strategy of MAC-Lite is to abandon retransmissions and pass partially corrupted packets to higher layers. Simulation results verified that UDP-Lite with MAC-Lite did improve the throughput and decreased the length of packet drop bursts for multimedia applications [Khayam et al., 2003a,b].

Lam and Liew [2004] mentioned another shortcoming of UDP-Lite: lack of flexibility for the application layer to handle corrupted packets. They proposed a new protocol which keeps the concept of UDP-Lite but has modifications that help to overcome some of its drawbacks. This new protocol is named UDP-Liter. It uses the traditional UDP header and does not change the checksum calculation. One major function of UDP-Liter is that it has a run-time option not to discard data when a checksum fails. The other function is corruption notification (CN). Each packet delivered by UDP-Liter has a CN through a parameter which can tell the application whether the packet is corrupted or not. With the CN, the application can handle normal packets and corrupted ones differently. Although UDP-Liter does not compensate for packets lost in transmission between devices, it saves a number of discarded packets and makes the data loss ratio decrease.

There are some other UDP based protocols intended to improve performance of real-time multimedia communication by reducing the ratio of discarded packets. CUDP

(complete UDP) was proposed by Zheng and Boyce [2001] for video transmission over internet-to-wireless networks. Instead of discarding error frames, CUDP passes them to applications with the information of the location of erroneous frames. The corrupted frame can be forced into an all "1" sequence so a media decoder can invoke error concealment when it get this special sequence. The effect of error frames can be minimized in this way. Zheng and Boyce [2001] also presented two FEC (forward error correction) coding schemes at the packet level that can be used in combination with CUDP. When employing FEC, CUDP erases corrupted frames and utilizes error-free frames within the same packet to recover the lost information. If the packet loss rate gets big due to network congestion or wireless channel errors, CUDP's ability of packet recovery shrinks. The advantage of CUDP also depends on the video quality requirement.

Forwarding corrupted packets to the application rather than discarding them is acceptable for certain types of real-time applications such as video and audio. More packets can be received by applications so that the overall high-level throughput is improved and the media quality is enhanced. For multimedia applications, many media decoders have the right to decide whether to use or discard the corrupted packet. However, it is a totally different story in NCSs. A damaged control signal may be harmful to the whole system and has to be rejected. Therefore, this type of protocol is not suitable for real-time control systems.

Relaying corrupted packets reduces the data discard ratio but cannot help if packets are dropped before they reach the destination. Especially, for a wireless network, the vulnerable wireless medium is more susceptible to errors and losses than a wired one. There is no doubt that retransmission is the most well known and effective way to provide reliable communications. TCP features its retransmission scheme but is not able to be employed as the transport layer protocol for real-time applications. Many researchers focus on developing protocols which provide retransmission when keeping data timeliness in mind for delay sensitive applications.

The reliable UDP (RUDP) protocol was designed at Bell Labs to support reliable

transport of telephony signaling [Bova and Krivoruchk, 1999]. It rides on top of UDP and adds some features that make packet delivery reliable and in-order. Acknowledgement of received data, windowing and congestion control, retransmission of lost packets, and overbuffering are four major characteristics of RUDP. According to RUDP's retransmission algorithm, the sender enables retransmission at the time-out of the retransmission timer or when it receives an EACK (extended ACK) segment. If an EACK segment is received, the sender checks this segment and determines which segments should be retransmitted. An EACK segment contains the ACK number and the last out-of-sequence ACK number. All segments between these two sequence numbers that are on the unacknowledged sent queue are to be retransmitted. In the condition of a retransmission time-out, all messages on the unacknowledged sent queue are retransmitted. The time-out value is configurable in a retransmission timer. The timer is initialized every time a segment is sent and will reset when receiving an acknowledgement. 600 milliseconds is the recommended time-out value. Working together with the retransmission timer is the retransmission counter. This counter maintains the number of times a segment has been retransmitted. There is a configurable maximum value for the counter. Once the counter exceeds its maximum, the virtual connection is considered broken and a connection auto reset will be performed.

RUDP employs a SYN segment to establish the connection and synchronize sequence numbers between two hosts. Both the value of the retransmission timer and the maximum number of the retransmission counter are specified in the SYN segment. These two values are defined before the build of the connection and they are constant over the connection's lifetime. This implies that RUDP does not take into account changes in the network's condition. When the network's condition turns bad, inappropriate retransmission will burden the link and cause a worse network condition, and lead to degradation of the overall performance of communication.

The dynamic retransmission control for RUDP was proposed by Le et al [2009]. Their enhanced RUDP (E-RUDP) enables dynamic retransmission by employing the data packet aging parameter, round-trip times (RTT) and the maximum cumulative

ACK count. Data packet aging specifies the maximum delay the application can tolerate. Calculations of retransmission timeout and maximum retransmission count are defined as below, where n is the max cumulative ACK count:

$$\begin{aligned} Retransmission\ Timeout &= \left(\frac{n-1}{2} + 1\right) \\ Max\ Retransmission\ Count &= \lceil \frac{Data\ Packet\ Aging}{Retransmission\ Timeout} \rceil \end{aligned}$$

Their simulation results show that E-RUDP offers the highest average throughput when compared with TCP, UDP and RUDP [Le et al., 2009]. However, E-RUDP is designed for a typical airborne network environment to improve video quality; further improvement to E-RUDP is needed to make it a more user friendly transport layer protocol.

RUDP and a RUDP based protocol such as E-RUDP work well for certain real-time applications whose emphasis is high throughput; however, they are not the most suitable protocols for real-time applications in NCSs. In NCSs, throughput is no longer the paramount element of system performance and data timeliness and reliability turn out to be more important. It is known that the predominantly periodic traffic pattern is a unique feature of industrial real-time control systems [Cena et al., 2008]. Under normal conditions, the control frequency is fixed and the traffic load of the control system is specified ahead. These characteristics mean that packets have fixed size and are sent in a fixed period corresponding to the control frequency. Effectiveness of a packet is very important. A packet turns ineffective when its control period ends and a new packet arrives. Out-of-date packets could make devices perform vital actions that lead to the failure of the whole system. A successfully received error-free packet may be rejected if it does not arrive in time. In real-time control systems, the transmission protocol should strictly direct both sender and receiver to discard a data packet if it is ineffective.

Some researchers have attempted to develop methods of providing reliable data

transfer service with a guarantee of data effectiveness. Jonsson and Kunert [2008] established a real-time scheduling analysis framework for real-time communication in a wireless network. This framework combines ARQ (Automatic Repeat-reQuest) with real-time scheduling analysis to achieve both reliability and real-time support. The defined ARQ protocol in this framework is rather straightforward as both congestion control and flow control is omitted. Retransmission is enabled as the technique to compensate for packet loss. The allowed number of retransmissions and the length of timeouts for retransmissions are calculated statically when they analyse the real-time timing and scheduling, therefore the retransmitted packets are ensured to be meaningful according to their deadlines. That is how both reliability and timeliness of transmission can be guaranteed. However, as timing and scheduling analysis are done before communication happens, the retransmission timeout duration and the number of retransmissions are predefined and will not change after traffic starts. Thus, these two values cannot be adjusted if the state of the network changes. It is well known that wireless channel is not as stable as a wired one and varies easily. This framework could not be suitable if the variation of channel is not taken into account. Another drawback of this framework is that it can only be adapted to single-hop networks. Further enhancements to this protocol are needed.

2.5 Other Network Elements that Influence Network Performance

In the previous sections, the reviewed methods for network performance improvement are either control methodologies or new network protocols. In fact, there are many other network elements, such as data rates, packet sizes and even the number of communication connections, which can affect the overall network performance. Research in this area has been limited to date.

The time delay between the application layers of the client and the remote control target under different sampling frequencies in an NCS has been measured and analysed

by Cheng et al [2007]. Experiments were conducted in two different scenarios: 1. an NCS including both the Ethernet and the CAN network (Controller Area Network); 2. an NCS using IEEE 802.11b technology with the CAN network. Their results show that different sampling rates and environments will greatly affect the delay. In scenario one, a lower sampling frequency will lead to a smaller mean delay around 2-3 ms, and the delay increases when the sampling frequency gets higher since the transmission over the network becomes busy. In scenario two, the delay cannot be solely determined by the sampling time. This is because the transmission efficiency in a wireless network is not as reliable as in a wired one. Cheng et al [2007] also indicate that the time-delay may become unacceptable in some cases where data transmission rates are high.

Lee et al [2002] tried to figure out the relation between the maximum transmission unit (MTU) and ad hoc network performance. In the case that all stations in the network stay stable or have small movements, the throughput rate increases as MTU does. Under bit error rate (BER)-base routing circumstances, MTU performs better if its size is between 250bytes and 750bytes. Performance of the simulated ad hoc network was found to reach its peak when MTU is set to 500bytes with 10-4 BER [Lee et al., 2002]. W. H. He and his colleagues gave a formal description of UDP performance in terms of end-to-end delays in ad hoc networks and theoretically analysed the optimal UDP packet size [He et al., 2007]. They defined and used the concept of maximum continuous throughput (MCT) [He et al., 2007]. It represents the maximum number of packets sent along the same route by one single continuous transmission. Several network scenarios with different values of MCT were set to analyse the performance of UDP with different packet sizes. Hop numbers was also considered as a factor in the analysis and it varied from 1 to 10. As the node's mobility and traffic load are random in real ad hoc networks, W. H. He proposed that the loose optimal range of packet size for UDP is between 400bytes and 800bytes [He et al., 2007].

Itaya [2004] observed fluctuation of transmission latency during UDP packet exchange in ad hoc wireless groups. Simulation results show that the faster the packet

generation rate gets, the larger is the fluctuation of packet inter-arrival times. In the low packet generation regime, the packet transmissions are stable with a minimal fluctuation. In the high rate regime, where saturation of throughput happens, the delays fluctuate wildly. The occurrence of large fluctuation is due to the intrinsic protocol dynamics. Instead of modifying intrinsic protocols to minimize the fluctuation, Itaya indicated that it is less complex to identify a rate regime in which the best trade-off between fluctuation and throughput can be found. For those real-time applications that prefer deterministic delays as well as relatively high throughput, a proper regime of packet generation rate is appreciated. He used VOIP applications as examples and applied his method for estimating the conditions of latency fluctuation to the design of this type of fast data exchange applications. Given the number of nodes doing full mutual communication in the network, Itaya can specify the suitable packet size and data rate to achieve the best network performance (high throughput with rather small latency fluctuations). John et al [2008] also believe that the influence of modes of data transmission on the efficiency of wireless networks is worthy of analysis. Several parameters were listed to help for the description of data transmission modes. They are the size of the sent files, the number of connections and the duration between the transmissions of the files.

For an IEEE 802.11 network, the primary source of packet losses is always considered to the errors in its physical medium. However, Salyers et al [2008] showed that the wireless device itself plays an important role in packet loss. They analysed occurrences of instantaneous loss and bursty loss for two receiving nodes respectively in an ad hoc network [Salyers et al., 2008]. Experimental results show that the distance between these two nodes and the data rate of packet reception influences the patterns of loss.

Wang et al [2006] presented an optimized deployment strategy of mobile agents in a wireless sensor networks which can efficiently decrease and stabilize the response time of the central service node to requests from sensor nodes and improve real-time performance. Parameters used in this deployment strategy are the creation sequence,

the priority and the energy consumption of mobile agents. These three key factors are synthesized to create an integrated metric which makes the deployment strategy user friendly.

2.6 Summary of the Literature Review

As we have seen, in an NCS, providing reliable real-time transmission with deterministic latency is the primary goal of a control network in order to guarantee high performance of the system. When some data networks, such as Ethernet and WLANs, are introduced to NCSs, for their advantages with respect to scalability, low cost and high capacity, the performance of the control system is degraded due to network-induced packet losses, non-deterministic delays and jitter. It is hard to model random transmission delays accurately with mathematical formulations and compensate for them through delay analysis in random access networks. This gets even harder in wireless networks as they have additional transmission characteristics compared with wired networks such as higher error rates, wireless link errors, host mobility, longer delays, larger numbers of dropped packets and lower bandwidth.

In this literature review, time-delay and packet loss issues in control systems with wireless networks were surveyed. Recent work on compensating for transmission latency and data loss were studied. Many control methodologies have been developed to make network-induced delays deterministic in NCSs. Such work is also conducted from the aspect of the communication network by designing new protocols, especially MAC layer protocols. One of the basic ideas for these MAC layer protocols is try to control the random channel access and guarantee earlier access for packets with higher timing priorities.

Research on reducing packet loss ratios in wireless networks was reviewed with the focus on UDP protocol modification. Many UDP based protocols have been proposed for real-time multimedia applications and their emphases are on the improvement of network throughput and video/audio quality. However, technical gaps still remain to

be filled. Some methodologies can only work well when the channel congestion is slight. Some protocols are incompatible with other protocols on the same layer and higher layers. And there are also some schemes which achieve better performance in delivering data timeliness but sacrificed the quality of other services such as the data loss rate.

The most recent research described in Section 2.5 shows that network performance can be affected by some other elements in a network apart from the network protocol. It can be regarded as a supplementary effort to improve network performance in an NCS while employing new control methodologies or network protocols.

The work presented in this thesis aims to fill the technical gaps which the literature cited above has not bridged yet. As guaranteeing reliable and timely transmission for real-time applications in a wireless NCS is a major challenge, a transport protocol that provides for reliable transmission of typical data packets and keeps data timeliness in mind is developed. Furthermore, strategies for network layout are also proposed as they can be good supplements to network protocols or control methodologies to achieve better overall network performance in an NCS.

Chapter 3

Network Layout Strategies for WNCSs

Wireless networks are being increasingly proposed as the basis for implementation of real-time NCSs. However, they introduce unpredictable transmission latencies and potential packet dropouts that can lead to system performance degradation or even cause system instability. Most of the literature reviewed in Chapter 2 presented new protocols or control methodologies to compensate for network introduced delays, jitter and data losses. Although network protocols or new controller designs can help reduce latency, jitter, or packet dropouts, they are not the only way to solve those network introduced problems in WNCSs.

Network performance, especially wireless network performance, is affected by many factors as a wireless channel is not as reliable as a wired one and is more vulnerable to errors. The papers reviewed in Section 2.5 showed that many elements in the network other than transmission protocols can be manipulated to improve network performance. In this chapter, we analyse, through simulations, how sensors' data rates and network layouts affect network performance in terms of delays and data losses. The data rate of sensors is one of the key factors that define traffic load in the network and can greatly influence network performance.

Sensors in an NCS may or may not have the same data rate. Two case studies were

conducted, with the first one for the case that all sensors in the NCS had the same data rate, while the sensors in the second case study had different data rates. We aimed to find a network layout that can help improve network performance for each case. Strategies are proposed to reduce communication delays and the packet dropout rate when sensors with different data rates are configured in one wireless network.

3.1 Basic Network Configuration and Evaluation Metrics in Simulations

The performance of data transmissions between multiple sensors and a central controller in a wireless NCS was tested via simulation studies. Simulations were conducted using the popular simulation tool, NS-2. A brief introduction to NS-2 can be found in Section 4.4. An IEEE802.11b network was modelled with the basic configuration listed in Table 3.1. The maximum range for a wireless network in NS-2 is 250*250 square metres, and the networks in our simulations were set up to be single-hop networks for simplicity. Since TCP traffic is not directly applicable for real-time communication, UDP was employed as the transport layer protocol in our investigation.

Table 3.1: Basic configuration for wireless model in simulations

Network standard	IEEE 802.11b
Radio channel data rate	1.0 Mbps
Network area	250m * 250m square
Radio-propagation model	Two-ray ground
Routing protocol	Dynamic Source Routing
Wireless interface (MAC) buffer type	Drop-tail priority queue
AntennaD	OmniAntenna

In the wireless NCS we simulated, all communicating sensors were placed at fixed locations within the network's transmission range. The real-time application in each sensor generated a data packet of 200bytes and transmitted it to the controller at the

sensor's data rate. We assumed that all sensors start their communications when the simulation starts. This means that sensors with different data rates may send packets at different times provided the rates are not whole multiples of one another while sensors with the same data rate may send a packet to the controller at the same time. Every scenario in our simulations was conducted for 5 seconds. For performance evaluation, the first 10 data packets was not counted because the nodes need some initial set-up time for routing which would otherwise skew the measurements. The evaluations performed were:

1. The relationship between the layout of the network, i.e., the network architecture, and the communication delay and packet dropout rate;

2. The relationship between the data rate of sensors, and the communication delay and jitter.

3.2 Case One: Sensors with the Same Data Rate

In this case study, the network's performance was examined when all sensors have the same data rate. Ten sensors communicate with the central controller in all scenarios. As the data rate and packet size are fixed for all sensors, we could concentrate on variations in the network layout to determine how to obtain the smallest transmission latencies and packet loss rates.

Ten sensors were deployed in five different ways with the controller in the centre, as shown in Table 3.2. These five different ways are named one layer, two layers, three layers, five layers and ten layers, where each "layer" consists of one or more sensors at the same distance from the controller. All the sensors are 100 metres away from the controller when they are in one layer. Five sensors are 120 metres away from the controller while the other 5 are 80 metres away from the controller in the situation of two layers. For the situation of three layers, the distance between the controller and sensors 1 to 3 is 70 metres; sensors 4 to 7 have a distance of 105 metres away from the controller while the rest of the sensors are 140 metres away. In the five-layer

setup, every two sensors have a certain distance to the controller: 60, 80,100, 120 and 140 metres. The setup of ten layers means each sensor has a different distance to the controller, ranging from 60 metres to 150 metres.

Table 3.2: Layers and distances between sensors and the controller

<table>
<tr><td rowspan="2">Sensor Number</td><td colspan="5">Distance between sensors and the controller (metres)</td></tr>
<tr><td>1 layer</td><td>2 layer</td><td>3 layer</td><td>5 layer</td><td>10 layer</td></tr>
<tr><td>1</td><td rowspan="10">100</td><td rowspan="5">80</td><td rowspan="3">70</td><td rowspan="2">60</td><td>60</td></tr>
<tr><td>2</td><td>70</td></tr>
<tr><td>3</td><td rowspan="2">80</td><td>80</td></tr>
<tr><td>4</td><td rowspan="4">105</td><td>90</td></tr>
<tr><td>5</td><td>100</td><td rowspan="2">100</td></tr>
<tr><td>6</td><td rowspan="4">120</td><td>110</td></tr>
<tr><td>7</td><td rowspan="2">120</td><td>120</td></tr>
<tr><td>8</td><td rowspan="3">140</td><td>130</td></tr>
<tr><td>9</td><td>140</td><td rowspan="2">140</td></tr>
<tr><td>10</td><td>150</td></tr>
</table>

After the five different layouts were set up, we generated different simulation scenarios with different values of data rates for all sensors. The data rate was changed from 80kbps to 120kbps. For each scenario, the average delay of all received data packets and the percentage of dropped packets was calculated. Tables 3.3 and 3.4 tabulate the simulation results.

Table 3.3 shows that the deployment of one layer produced the worst result in most cases in terms of the end-to-end delay. This is not surprising because this layout implies that the interference between sensors is the most severe. As all the sensors have the same data rate, they attempt to send packets to the controller from the same distance and at the same time. To avoid such strong interference, sensors should be put at different distances to the controller (or, equivalently, they should all have different fixed offsets from their common period).

Indeed, the results shown in Table 3.3 confirm that the 3-layer and 10-layer settings perform better. However, it is still not clear which setting works best. For example, the smallest average delay (8.84365 milliseconds) is obtained if sensors are deployed into

three layers when the data rate of sensors is 90kbps. If the data rate grows to 100kbps, the minimum value of the average transmission latency is recorded in the case where sensors are put into ten layers. Interference between sensors is thus the dominant cause of transmission delays, rather than the physical distance, when the number of sensors is large. Consequently, we can conclude that there is no preferred way of placing sensors in order to get the smallest delay, given the short physical distances suitable for wireless transmissions. Since the maximum transmission range for wireless sensors is only 250 metres, the propagation delay is insignificant compared to interference from other sensors. Nevertheless, the results still confirm our intuition that the worst case is to put all sensors at the same distance from the controller, with the same data rate.

Table 3.3: Average end-to-end delays

Data rate	Average end-to-end delay (milliseconds)				
(kbps)	1 layer	2 layer	3 layer	5 layer	10 layer
80	8.83599	8.67104	8.73761	8.7821	8.75565
90	9.8784	9.20081	8.84365	8.82675	9.0127
95	11.382	11.361	13.807	10.235	12.556
96.97	32.8284	25.701	16.0296	20.0477	18.7519
100	95.7255	92.7384	89.43	102.765	72.4401
120	437.389	434.006	442.711	448.842	385.193

Table 3.4: Data packet loss ratio

Data rate	Percentage of dropped data packets in communications)				
(kbps)	1 layer	2 layer	3 layer	5 layer	10 layer
80 – 95	0%	0%	0%	0%	0%
96.97	0.56%	0%	0.07%	0.1%	0.03%
100	6.36%	6.43%	5.94%	6.33%	5.3%
120	19.0%	19.1%	19.0%	19.1%	18.3%

It can be observed from Table 3.3 that the transmission delay keeps growing statistically as the data rate gets higher. When the data rate increases to a certain value, the time delay exceeds the transmission interval (period), which results in a packet

"dropout", i.e., a packet which arrives too late to be useful to the real-time computation, or not at all. Once the data rate exceeds this critical value, the delay increases dramatically and a large number of dropouts can be seen. The percentage of dropped packets to the total number of data packets transferred in the communication is listed in Table 3.4 according to different data rates.

For simulations in this case study, with 10 sensors, the critical data rate is approximately 96.97kbps (which means each sensor sends a data packet of 200 bytes every 16.5 milliseconds). At this point, the average delay begins to exceed the transmission period and dropped packets begin to occur. Only when sensors are placed into three layers can the average delay (16.03 milliseconds) be controlled within a transmission interval (16.5 milliseconds). If the data rate is less than 96.97kbps, no dropouts will happen, as seen in the first row of Table 3.4. However, when the data rate reaches 100kbps, the average delay is about 4 times more than the transmission period and the percentage of dropouts ranges from 5.3% to 6.43%. The results get even worse if the data rate reaches 120kbps, and even the smallest average delay could be 30 times as long as the transmission period, besides which, more than 18.26% data packets are lost. This is because packets are taking longer to reach their destinations than a transmission interval, so there is a cumulative backlog of packets. Obviously, this is not acceptable in real-time control. Thus the data rate must be carefully chosen to make sure that almost every data packet can be received in a control period.

As a conclusion for case study one, we provide the following solutions for reducing transmission delays and the packet dropout rate when all sensors in an IEEE802.11b network have the same data rate:

- The data rate must be chosen with the maximum number of competing sensors in mind; and
- Although the overall sensor layout is not a dominant factor in overall network performance, sensors should not all be placed at the same distance to the controller.

3.3 Case Two: Sensors with Different Data Rates

The previous case study was carried out without variations in data rate, i.e. all sensors had the same data rate. Now we model and simulate the scenarios in which sensors have different data rates. It is assumed that two different data rates exist in the network, and the sensors are divided into two groups according to their data rates. Group 1 is composed of the sensors with a higher data rate, while group 2 consists of the sensors with a relatively low one. Both groups have the same number of sensors, and all sensors communicate with the central controller. The packet size is still 200 bytes which is the same for all sensors. Again, the simulation time of each run was carried out for 5 seconds, during which all stations stayed still without moving.

With the experiences obtained from case study one, we carefully chose data rates for the sensors to guarantee that there was no severe transmission competition between sensors. The total number of sensors was changed from 4 to 30 to generate different scenarios. In each scenario, the average end-to-end delay of all successfully received data packets was calculated in three situations:

1. Group 2 (composed of slow sensors) put closer to the controller;

2. Both groups placed at the same distance to the controller;

3. Group 2 (composed of slow sensors) deployed further away from the controller.

The delay performances are summarized in Figures 3.1 through 3.5. The dotted bar represents the average delay in situation 1, while the lined bar records the average delay in situation 2. The average delay in situation 3 is shown by the blank bar. As shown in Figure 3.1, when there are only a few sensors, such as four, the best delay performance is obtained when group 2, which has the lower data rate, is closer to the controller. This happens because there is little interference between the sensors and their distance is still the main cause of packet transmission latency. The closer all sensors are to the controller, the smaller are the delays of packet transmissions. However, as the number of sensors increases, the results change significantly.

Figures 3.2 and 3.3 show that when there are 8 or 10 sensors, the shortest average delay is recorded if group 2, with the lower data rate, is put further away from the controller. The longest average delay occurs if group 2 is deployed closer to the controller.

As shown in Figures 3.4 and 3.5, the smallest average delay can still be obtained by deploying the sensors of group 2 at a further distance from controller. For 30 sensors, the latency decreases significantly (by 5 milliseconds) if we put group 2 further away rather than in the same area as group 1. The main difference in the results shown in Figures 3.4 and 3.5, compared to those in Figures 3.2 and 3.3, is that the worst case happens when the two groups are at the same distance to the controller.

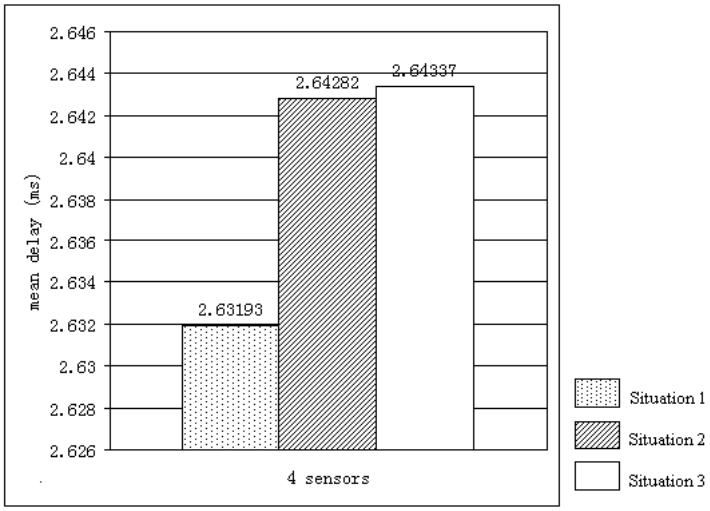

Figure 3.1: Average delays when there are 4 sensors

Scenarios with 30 sensors are analysed in detail below. Group 1 consists of sensors 1 to 15. Sensors 16 to 30 are in group 2. Data rates were set to relatively small values to reduce the interference among sensors as there were 30 sensors in these scenarios. Sensors in group 1 had a data rate of 32kbps while sensors in group 2 had a data rate of 20kbps. Three sub-scenarios were simulated according to the three situations mentioned above. In scenario 1, group 2 is 25 metres away from the controller. Distances between the controller and the sensors in group 2 are 50 metres in scenario 2 and 80 metres in scenario 3, respectively. Group 1 was always 50 metres away from

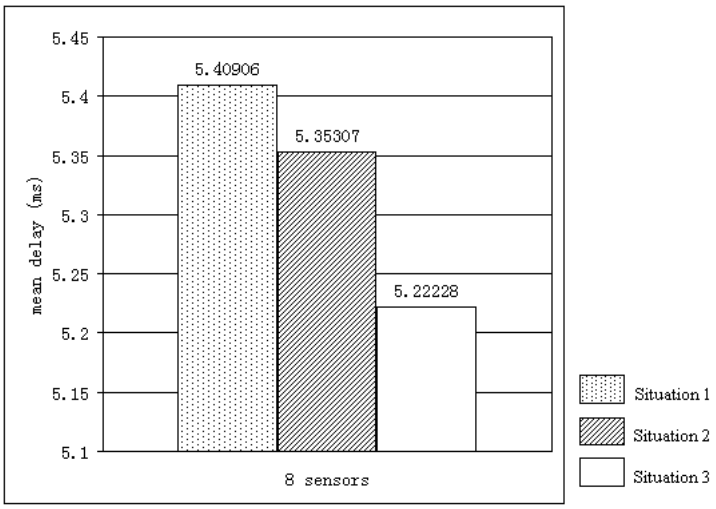

Figure 3.2: Average delays when there are 8 sensors

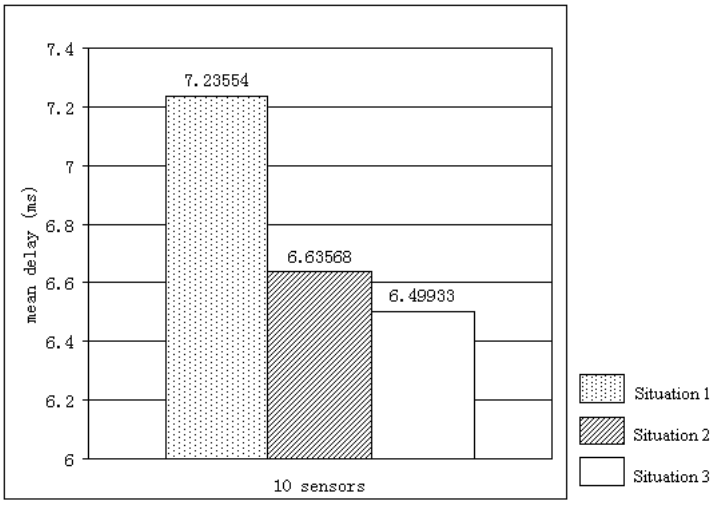

Figure 3.3: Average delays when there are 10 sensors

the controller in these scenarios. Therefore, the slow sensors were put closer to the controller than the fast ones in scenario 1. In scenario 2, all sensors were at the same distance to the controller, while fast sensors were deployed closer to the controller than slow ones in the last scenario.

Table 3.5 lists the average delays of all successfully received data packets and the percentages of dropouts in the three scenarios. Not only is the percentage of dropouts

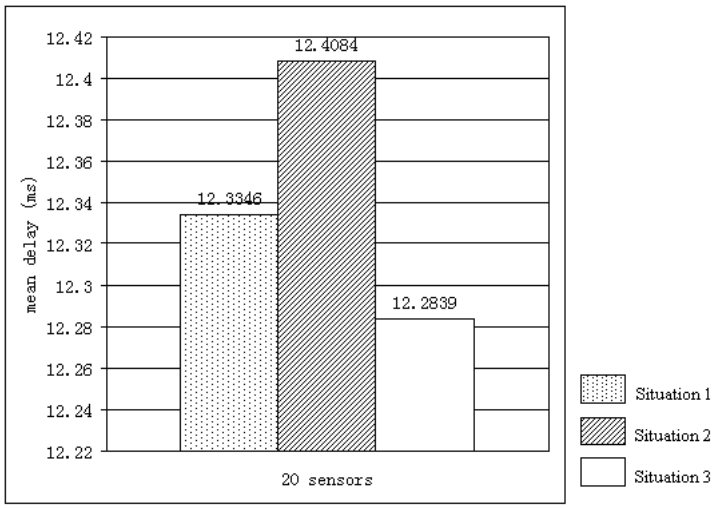

Figure 3.4: Average delays when there are 20 sensors

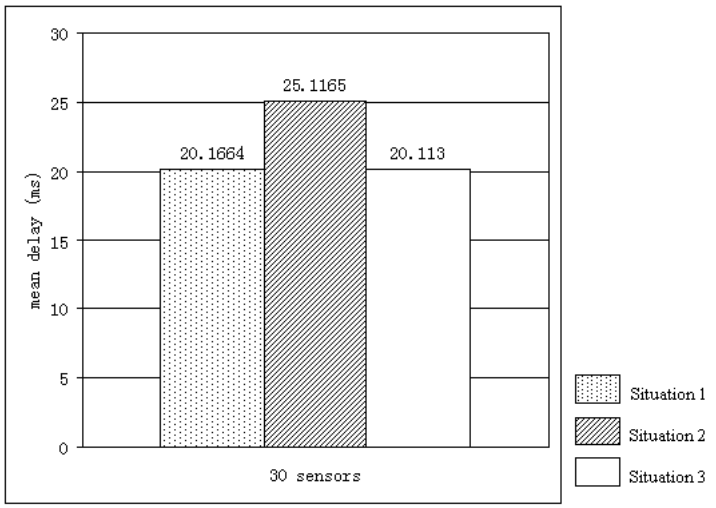

Figure 3.5: Average delays when there are 30 sensors

in scenario 3 three and four times smaller than those in scenarios 1 and 2, but also the average end-to-end delay in scenario 3 is 5 milliseconds shorter than the latency in scenario 2.

Sensor 6 and sensor 22 are chosen as the representatives of their groups and the transmission behaviours of these two sensors can be seen in Figures 3.6 and 3.7. Figure 3.6 depicts the end-to-end delay of every packet sent by sensor 6 (fast sensor) in scenario 2 and 3. The delay performance of sensor 22 (slow sensor) is described in

Table 3.5: Average delays and data loss rates in the case of 30 sensors

Sub scenario	Average delay (milliseconds)	Percentage of dropout
1	20.1664	0.12%
2	25.1165	0.16%
3	20.1130	0.04%

Figure 3.7. It was found that both the overall delay and jitter for the fast sensor and the slow sensor decreased in scenario 3.

- For sensor 6, the average delay of all its data packets successfully received at the controller is 26.498 milliseconds in scenario 2 and 17.945 milliseconds in scenario 3, respectively. Also the jitter was reduced from 215.599 milliseconds (scenario 2) to 73.0385 milliseconds (scenario 3).

- For sensor 22, the average end-to-end delay (jitter) is 27.0397 milliseconds (120.546 milliseconds) in scenario 2 and 21.9447 milliseconds (75.448 milliseconds) in scenario 3, respectively.

The reduction of the communication delay allows us to allocate more time for control computing, and the decrease in jitter enhances system stability and predictable timing. Thus, according to these simulation results, the basic strategies for reducing the time delay as well as jitter when sensors have different data rates are:

- Slow sensors should be placed farthest from the controller if there are a large number of sensors in the wireless network;and

- Faster sensors should be put nearer to the controller.

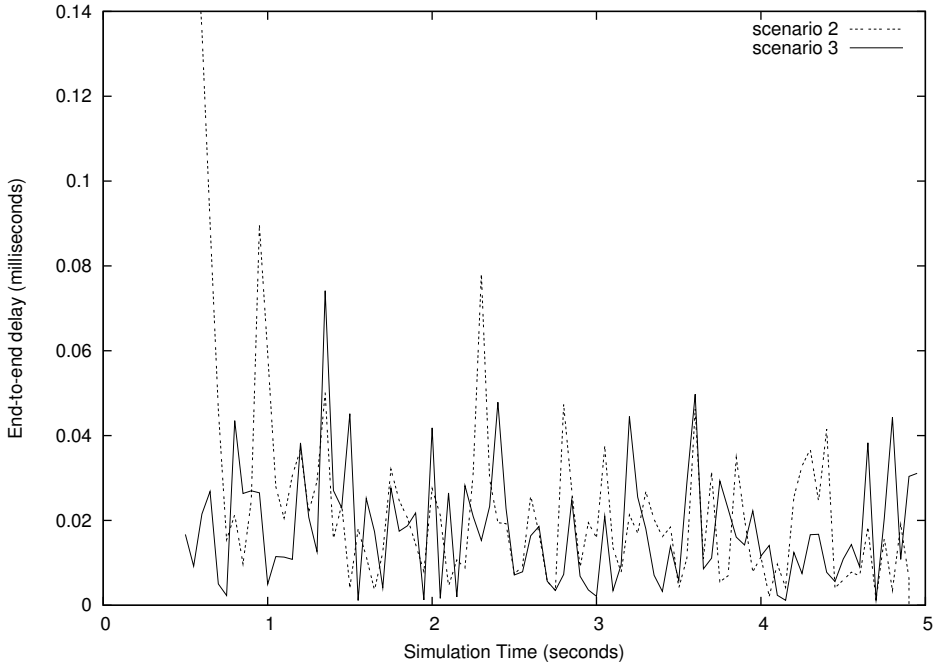

Figure 3.6: Delays of data packets sent by sensor 6 in scenarios 2 and 3

3.4 Chapter Summary

NCSs suffer from long and random delays and data losses due to the introduction of new networking technologies to control systems. Developing new network protocols and control methodologies are two major ways of dealing with these network-introduced problems. However, there also exist other methods for compensating for end-to-end delays, jitters and packet dropout rates in an NCS. In this chapter, some network layout strategies for IEEE 802.11b-based UDP/IP NCSs were developed based on simulation analyses. End-to-end delays, jitters and data loss rates can be significantly reduced by employing these strategies in the deployment of sensors.

Different sensors may have different data rates. In an NCS, there may be fast sensors and slow sensors used together. Our strategies for reducing delays as well as jitter in the case of sensors with different data rates are: to put faster sensors nearer

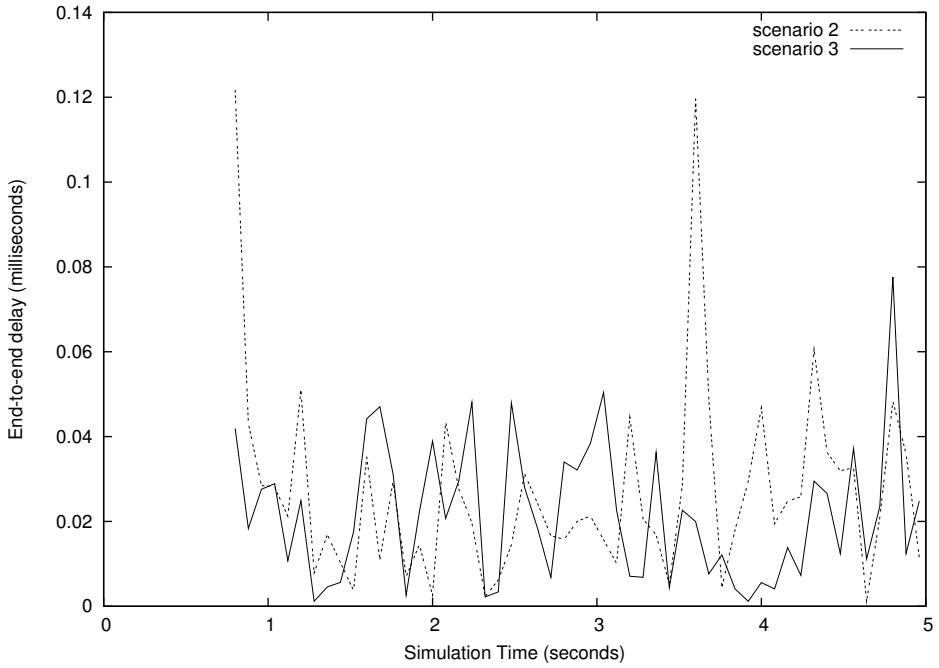

Figure 3.7: Delays of data packets sent by sensor 22 in scenarios 2 and 3

to the controller while slower sensors are placed further away from the controller. However, when all sensors have the same data rate, interference between sensors may be a dominant factor in overall network performance. In this case, the data rate must be chosen with the maximum number of competing sensors in mind, and sensors should not all be placed at the same distance to the controller.

Strategies in this chapter are proposed from the perspective of network deployment to improve real-time performance. From the aspect of communication scheme, a new transport protocol is designed and presented in the following chapter which is more efficient to provide reliable data transmission while guaranteeing data timeliness.

Chapter 4

Design of the Conditional Retransmission Enabled Transport Protocol

Providing reliable packet transmission as well as keeping data timeliness is a challenging problem in real-time wireless NCS research and development. The connectionless UDP protocol transmits data at high speed and is usually chosen as the transport layer protocol for real-time applications instead of TCP. However, reliable packet transfer is not guaranteed due to UDP's unacknowledged transmission. It is important to make modifications to UDP to improve its reliability but keep its fast transmission performance over wireless networks for real-time applications.

A transport layer protocol solving packet dropout related problem in wireless NCSs is developed in this research. This Conditional Retransmission Enabled Transport Protocol (CRETP) is based on modifications to UDP. It adds functions for retransmission and acknowledgment. The CRETP significantly improves transmission reliability while maintaining average delay at an acceptable level.

4.1 Logical Design of CRETP

Fast transmission enabled protocols such as UDP usually lack reliability guarantees. Protocols providing reliable and ordered data transfer, like TCP, often introduce long and non-deterministic transmission latencies as they have to make sure that every packet is correctly received by its destination. For those real-time systems requiring both timeliness and reliability of data transmission, a trade-off between timeliness and reliability must be considered.

The Conditional Retransmission Enabled Transport Protocol is a UDP based protocol with the ability to conditionally retransmit unacknowledged packets. Figure 4.1 shows the relationship of CRETP to the other protocols and layers of the TCP/IP protocol suite. CRETP lies between the application layer and the IP layer and, like UDP, serves as the intermediary between the application programs and the network operations.

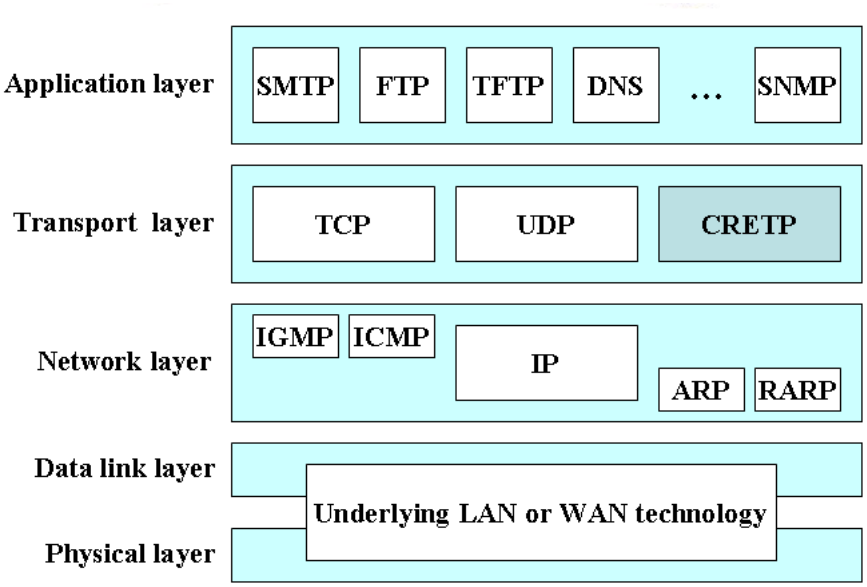

Figure 4.1: Position of CRETP in the TCP/IP protocol suite

CRETP enables a reliable transmission guarantee on the transport layer. Since this protocol is designed for NCSs, the reliable transfer of a packet not only means the

successful receipt of a packet at the receiving node, it also implies that the received packet is effective. By "effective" we mean a data packet in a real-time control system which satisfies both data correctness and data timeliness. A packet is considered effective only if: 1. it does not fail the checksum; and 2. it is not out-of-date. A data packet which fails its checksum is assumed to have been corrupted during transmission. When a data packet reaches its destination after the arrival of fresher data sent by the same source, it is recognized as an out-of-date packet. This could happen because packets transmitted in a connectionless way are independent and may choose different transmission routes. (Also see Section 4.3.1)

Our philosophy in the development of CRETP was based on the following aspects:

1. Keep the simplicity of UDP unchanged to maintain a similar level of fast transmission;

2. Enable a conditional retransmission function to provide reliable transfer of packets. Retransmission in CRETP is dynamic and employs parameters such as the current packet transmission latency in the network channel;

3. Check effectiveness of received packets to guarantee data timeliness.

Some feature services of CRETP that implement the above ideas are elaborated in the next few sections.

4.1.1 Connectionless Services with Acknowledgment

CRETP inherits UDP's connectionless service to the keep protocol simple. This means that there is no need to establish a connection with three-way handshaking and each packet is independent and can travel on a different path. There is no flow control in CRETP either, and hence no window mechanism. These simplicities save network resources and enable the fast data transmission of CRETP.

In order to provide reliable data transmission in an NCS, CRETP adds a sequence number to each data packet and enables acknowledgment for every successful data

transmission. Sequence numbers represent the sending time order of a sequence of control packets. The earlier the control packet is generated, the smaller is its sequence number. For each control packet, there should be a corresponding acknowledgment packet for its successful transmission. Figure 4.2 demonstrates the successful delivery of data packet and the receipt of its acknowledgment.

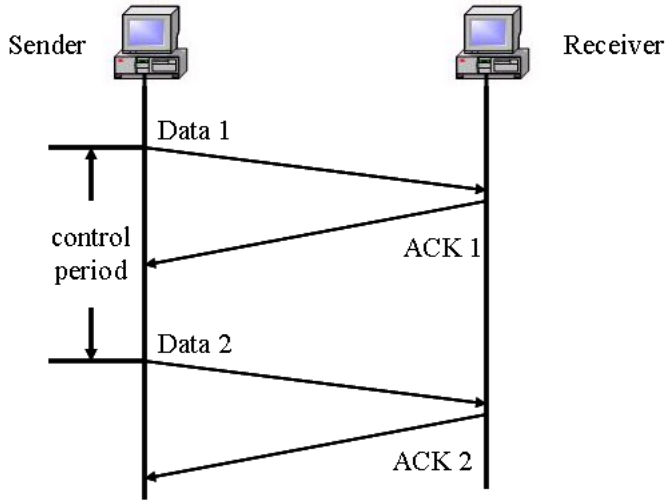

Figure 4.2: CRETP's delivery of data packet and its acknowledgment

4.1.2 Conditional Retransmission Service

Retransmission is the method used by CRETP to guarantee reliable data transfer. A packet will be retransmitted if it is lost during transmission or discarded by the receiver. Retransmission does not happen if a corresponding ACK of a data packet is received in time at the sender. CRETP retransmits a data packet in the following situations:

- A data packet is lost during transmission.
- A data packet is discarded at the receiving end due to its ineffectiveness.
- The corresponding ACK is lost during transmission.
- The corresponding ACK arrives late at the sender (i.e., the retransmission timer on the sending end expires before the corresponding ACK arrives).

Figures 4.3 to 4.6 illustrate retransmissions due to different reasons.

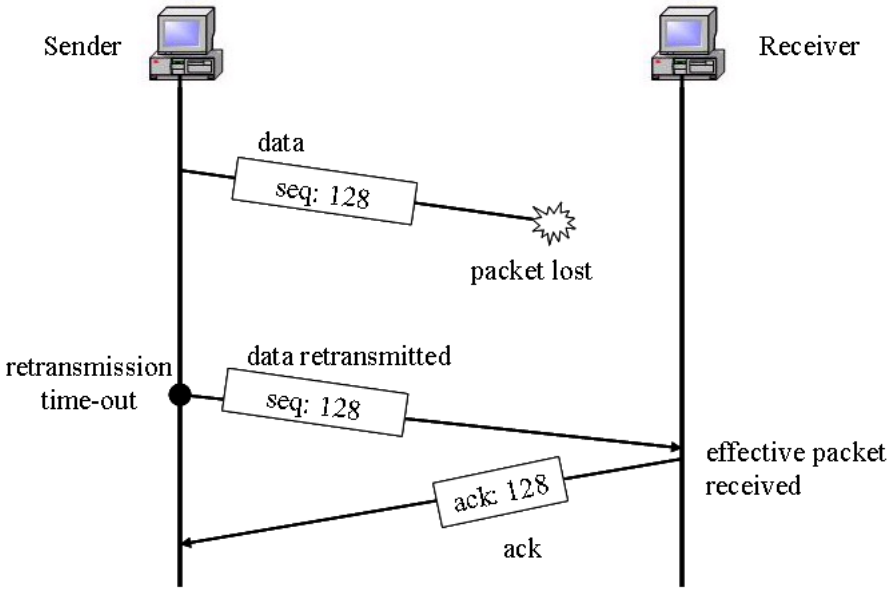

Figure 4.3: Retransmission of a lost packet

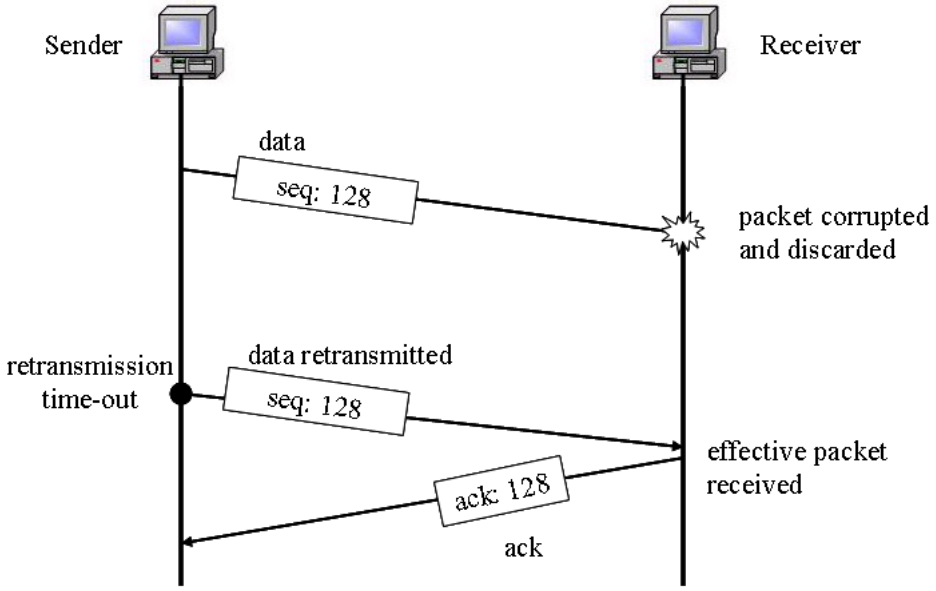

Figure 4.4: Retransmission of a corrupted packet

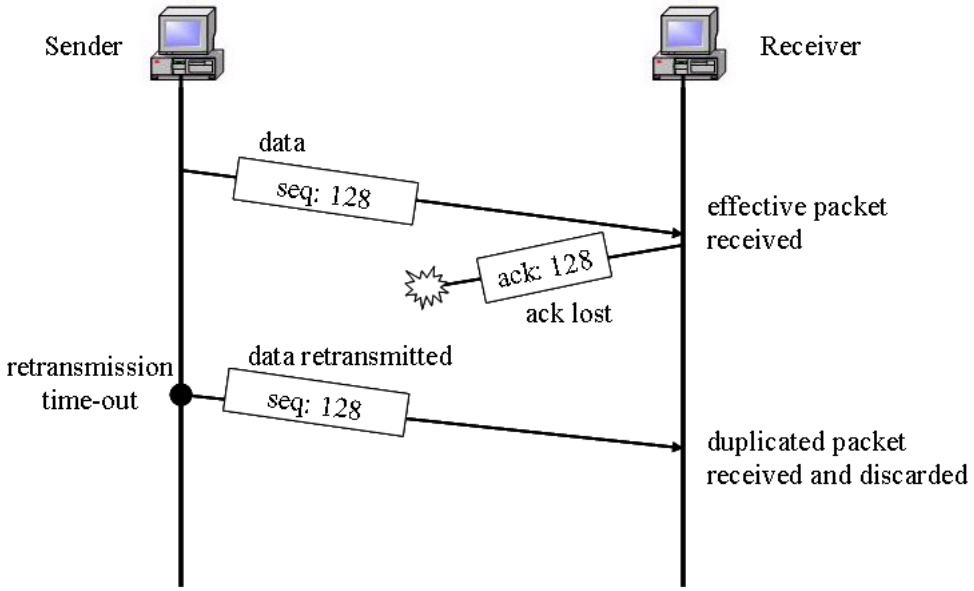

Figure 4.5: Retransmission due to a lost ACK

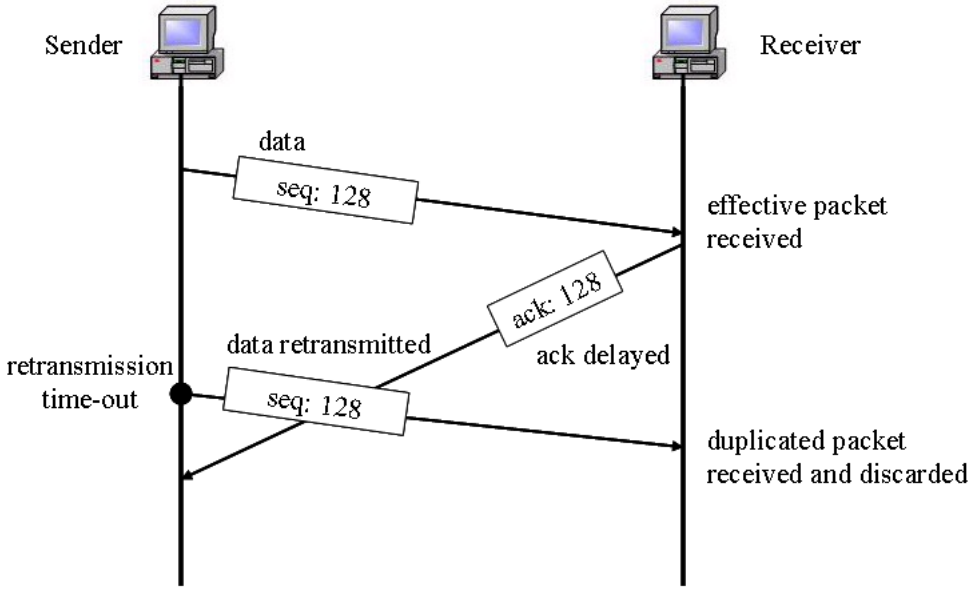

Figure 4.6: Retransmission due to a delayed ACK

However, CRETP's packet retransmission is different from TCP's endless by repeating retransmission when an ACK is not received by the sending end. The occurrence of retransmission is constrained as CRETP is designed for real-time NCSs. In

an NCS, a control packet is only effective within its control period. New information will be generated when the next control period starts. It is useless and a waste of network resources to retransmit a control packet when newer control data is ready to be sent. Therefore, CRETP stops retransmitting a packet if the current control period expires and new control data becomes available. A small number of retransmissions can achieve a notably lower data drop rate without producing too much redundant traffic.

4.1.3 Detection of Ineffective Data Packets

CRETP should have the ability to check the effectiveness of a received packet. Only an effective packet will be relayed to the upper layer and the destination CRETP will silently discard a packet that is considered ineffective. There are three types of ineffective packets: out-of-date packets, duplicated packets and packets which fail CRETP's checksum.

If a control packet arrives later than a newer one, the old packet is out-of-date. As mentioned in the previous section, a control packet is useless and ineffective if it is out-of-date. Duplicated data packets are transmitted by the source CRETP due to the loss of an ACK. Errors may happen during data transmission, and CRETP's checksum has the ability to detect a corrupted packet. These three types of packets will be refused by the destination CRETP.

4.1.4 State Transitions

To keep track of all the different events happening during data transmission, CRETP software can be implemented as a finite state machine. Table 4.1 shows the states for implementing CRETP.

CRETP progresses from one state to another in response to events. Figures 4.7 and 4.8 illustrate the state transitions of a source CRETP and a destination CRETP network node respectively.

In the source CRETP state transition diagram, the source can be in one of the

Table 4.1: States for CRETP

State	Description
CLOSED	Protocol is idle
WAIT	The source CRETP is waiting for data from the application layer
LISTEN	The destination CRETP is waiting for data from the source
PKT-GOT	CRETP has data from the application layer to send to its destination
PKT-SENT	Waiting for acknowledgment of data sent
PKT-RCVD	A CRETP packet has been received; checking effectiveness of the packet
PKT-PSD	Effective data is passed to the application layer

following states: CLOSED, WAIT, PKT-GOT and PKT-SENT. The explanation of Figure 4.7 is as follows.

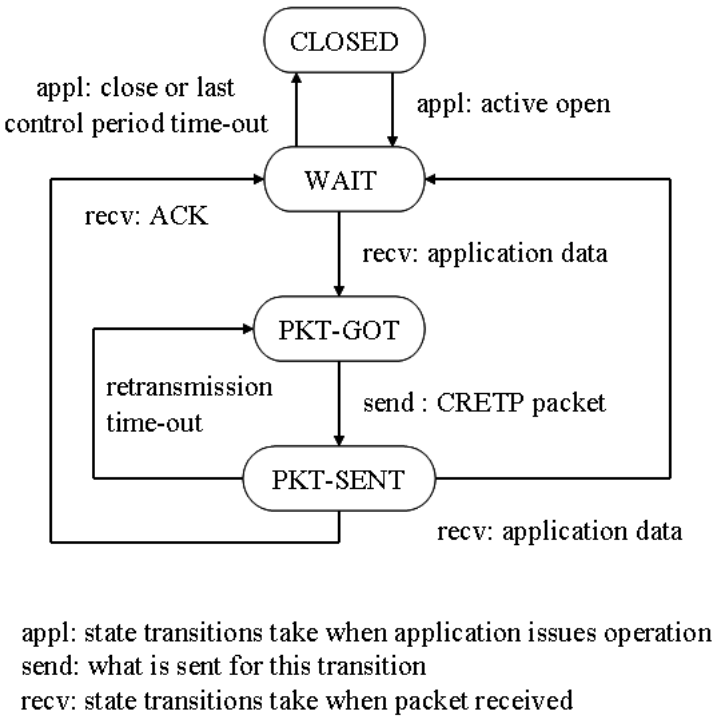

Figure 4.7: Source CRETP state transition diagram

- The source CRETP starts in the CLOSED state. While in this state, the source CRETP can receive an active open request from the source application program and goes to the WAIT state;

- While in the WAIT state, the source CRETP waits for data from the application layer. Once the application data arrives, CRETP goes to the PKT-GOT state. If CRETP is informed that the current data is the last one for this communication between sender and receiver, it will returns to the CLOSED state when the last control period expires;

- While in the PKT-GOT state, the data from the application layer is encapsulated in a CRETP packet. Then, the source CRETP sends this packet to its destination and goes to the PKT-SENT state. Note that this might be a retransmission of the previous packet due to the time-out of the retransmission timer;

- In the PKT-SENT state, the source CRETP checks if there is any new data coming from the application layer. Once new data arrives, the source CRETP cancels the retransmission timer and goes back to the WAIT state directly. In the case that there is no new data to be sent, CRETP may have the following actions. It starts a retransmission timer and waits for the acknowledgment. If an effective ACK is received before the time-out of the retransmission timer, CRETP considers the previous data transmission to be successful. Then it cancels the retransmission timer and returns to the WAIT state. If no ACK arrives and the timer expires, the source CRETP resets the retransmission timer and goes back to the PK-SENT state to retransmit the lost packet.

In the destination CRETP state transition diagram, the destination can be in one of the following states: CLOSED, LISTEN, PKT-RCVD and PKT-PST. The explanation of Figure 4.8 is as follows.

- The destination CRETP starts in the CLOSED state. While in this state, the CRETP can receive a passive open request from the application layer. It then goes to the LISTEN state;

- While in the LISTEN state, the destination CRETP may receive packets from the source CRETP. Once the packet arrives, CRETP goes to the PKT-RCVD state to check the effectiveness of the received data;
- In the PKT-RCVD state, the destination CRETP can have two different actions. If the received data is not effective, CRETP simply discards it and returns to the LISTEN state. Otherwise, CRETP transmits this packet to the application layer for further processing and goes to the PKT-PST state;
- While in the PKT-PST state, an ACK is prepared to send to the source CRETP. If the received packet is the last one of this communication sequence between the sender and receiver, a LAST ACK is sent, and the destination CRETP returns to the CLOSED state to end the communication. Otherwise, a normal ACK is sent and CRETP goes to the LISTEN state to continue receiving data.

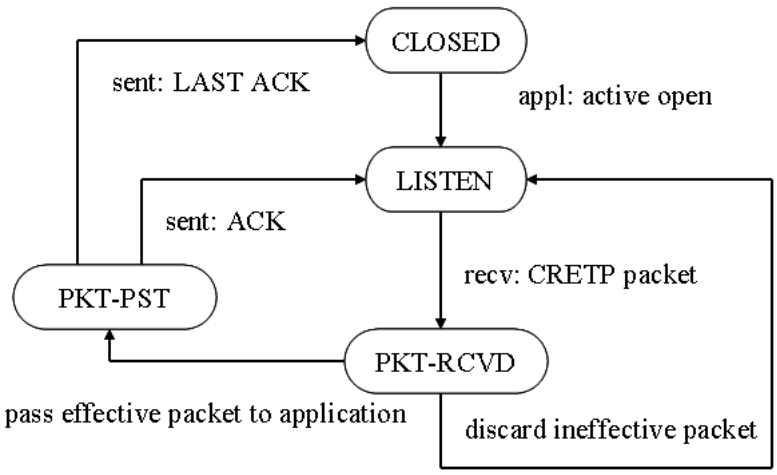

Figure 4.8: Destination CRETP state transition diagram

4.2 CRETP Packet Format

A CRETP packet has a fixed-size header of 16 bytes. Its header is an extension of the traditional UDP header. It can be seen in Figure 4.9 that CRETP keeps the first 8 bytes of its header the same as a UDP header. The following part of a CRETP header is divided into three fields. They are the sequence number field, the flag field and the timestamp field. The fields in a CRETP header are elaborated as follows:

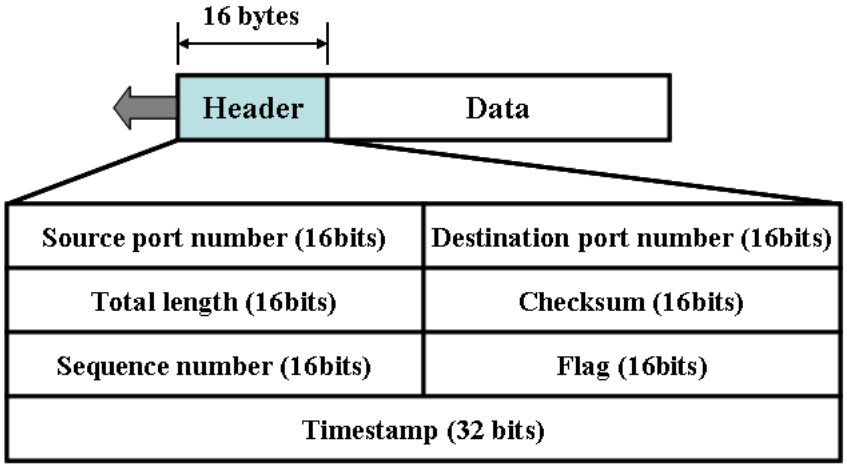

Figure 4.9: Format of a CRETP packet

- Source port number. This is the port number used by the process running on the source host. It is 16 bits long, which means that it can range from 0 to 65535.
- Destination port number. This is the port number used by the process running on the destination host. It has the same length as the source port number field.
- Total length. This is a 16-bit field that defines the length of the CRETP packet, header plus data. The minimum length is 16 bytes, which indicates a CRETP packet with only a header and no data.
- Checksum. This field is used to detect transmission errors over the entire CRETP packet (header plus data). The CRETP checksum calculation is exactly the same

as UDP's. When the receiver detects an error through the checksum, the packet is silently discarded.

- Sequence number. This 16-bit field contains the sequence number of a packet. Once a source CRETP is called by an application to start a series of data transfers, it will create a sequence number for each packet according to the incoming sequence. The earlier a packet comes, the smaller is its sequence number. Thus, the sequence number can show the age of a packet in a certain communication. In an NCS, the control packets are often fixed in size and are typically short, e.g., a few hundreds or even tens of bytes. The size of a CRETP data field is more than enough for a control packet. Therefore, each control packet can be encapsulated in one CRETP packet and gets a unique sequence number for the current communication between two nodes in an NCS. CRETP can handle the situation when a data packet is too big for a single CRETP packet. It divides the data packet into small segments and encapsulates these segments into different CRETP packets. However, this situation seldom happens in an NCS. For a CRETP ACK packet, the sequence number field records the sequence number of the data packet this ACK acknowledges.

- Flag. The flag field is provided to identify the type of a CRETP packet because CRETP uses two different kinds of packets. The value of a flag can only be 0 or 1. If the packet is a CRETP ACK message, the flag is 1. Otherwise, the flag is set to be 0 to denote a data packet.

- Timestamp. The timestamp is the last field in a CRETP header and it is 32 bits long. For a data packet, the timestamp records the time when this packet is sent. For a CRETP ACK, this field records the sending time of the data packet which this ACK corresponds to.

4.3 The Mechanisms in CRETP

CRETP is a reliable transport layer protocol, which means that an application program that delivers data to CRETP relies on CRETP to transmit those data to the application program on the other end without error, and without any packet loss or duplication. For real-time applications, there is an additional requirement for the arrival of packet at the destination. The packet should be time effective.

CRETP provides reliability using error control. Error control includes mechanisms for detecting corrupted packets, lost packets, out-of-date packets, and duplicated packets. Error detection and error correction are two aspects of error control.

Error detection in CRETP is achieved through the use of three methods: checksums, acknowledgments, and time-outs. Each CRETP packet has the checksum field in its header, which is used to check for corruption in the whole packet (header plus data). If a packet is corrupted, it is discarded. The destination CRETP also discards packets that are out-of-date no matter whether the packet is corrupted or not. Both corrupted packets and out-of-date ones are considered as ineffective packets. CRETP uses the acknowledgment method to confirm the receipt of those effective packets that have arrived at the destination. If a packet is not acknowledged before the retransmission time-out, it is considered to be either ineffective or lost.

Error correction in CRETP is conducted using a conditional retransmission mechanism. The source CRETP starts the retransmission of a packet if the ACK of this packet is not received before the retransmission timer expires. Mechanisms used in CRETP for error detection and error correction are elaborated in the following sections.

4.3.1 Mechanism for Data Effectiveness Detection

Not all the packets that reach the receiver will be accepted by the destination CRETP node at the transport layer. Effectiveness of an arrived packet is detected before CRETP conducts further actions.

There are several processes in the data effectiveness detection. Firstly, the received

packet cannot fail the CRETP checksum. This error detection method is used by most TCP/IP protocols. The checksum protects against corruption that may occur during the transmission of a packet. A packet that fails the checksum is considered as a corrupted packet and will be refused by CRETP. The CRETP checksum is exactly the same as the UDP checksum. Please refer to the UDP checksum if detailed specification is needed [Forouzan and Fegan, 2003]. Figure 4.4 demonstrates the situation when the receiver gets a corrupted packet.

When a packet successfully goes through the checksum, the destination CRETP checks its sequence number. In an NCS, data timeliness is an important element that can affect the overall system's performance. An expired packet has to be detected and rejected even if it does not fail the checksum. The sequence number in a data packet header is the identifier that tells the age of the data. The smaller the sequence number is, the older is the packet. CRETP compares sequence numbers of the current received packet with the value in its sequence number identifier. The sequence identifier is a variable which saves the sequence number of the latest effective packet. If the current received packet has a smaller sequence number, it is regarded as out-of-date and will be discarded. Figure 4.10 describes the occurrence of an out-of-date packet.

A duplicated packet is also ineffective and should not be relayed to the application layer. If the sequence number of the current received packet equals the value in the sequence identifier, this packet is defined as a duplicated one. CRETP simply discards duplicated packets. The arrival of a duplicated packet is illustrated in Figures 4.5 and 4.6, while Figure 4.11 is the flow chart for CRETP's mechanism for data effectiveness detection.

4.3.2 Acknowledgment Mechanism

CRETP uses the acknowledgment method to confirm the arrival of an effective data packet at the receiving end. The destination CRETP generates a corresponding ACK packet for each effective data packet it receives. Usually, a CRETP ACK is a CRETP packet with zero bytes of data. CRETP tells the type of a CRETP packet by checking

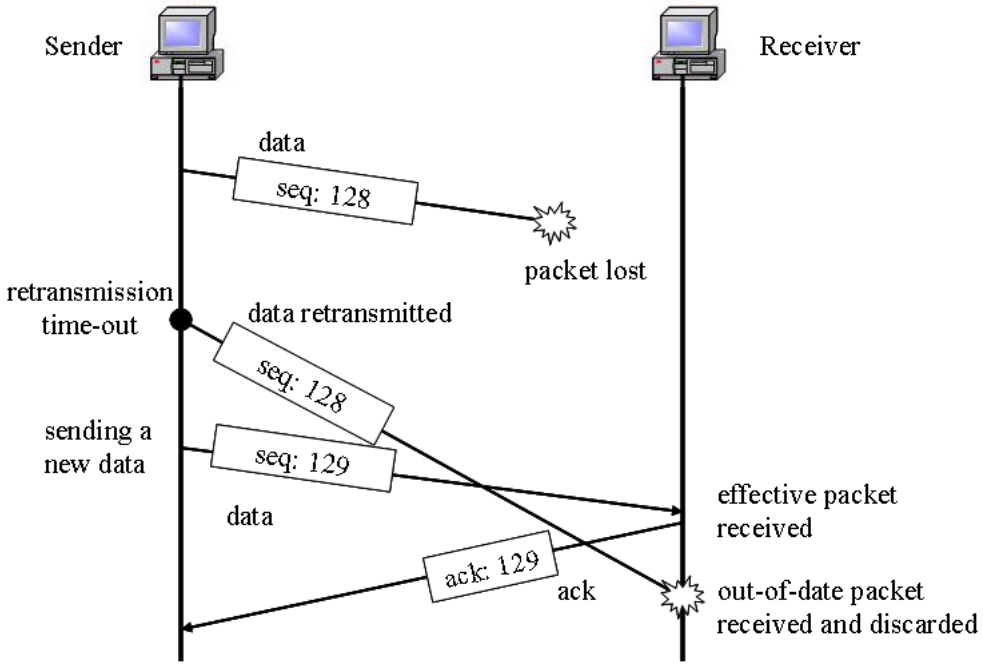

Figure 4.10: Occurrence of an out-of-date packet

the flag field in the packet header. When the flag equals 1, it indicates that this packet is a CRETP ACK.

Despite the flag field, there are other two fields in the ACK header that play important roles in CRETP. They are the sequence number field and the timestamp field. CRETP does not specify sequence numbers for its ACKs. When generating an ACK packet, CRETP saves the sequence number of the corresponding data packet into the ACK's sequence number field. By doing so, the source CRETP can tell which data packet this ACK acknowledges. The timestamp field in the ACK header provides key information for RTT calculation at the source CRETP. When an effective data packet is accepted by the destination CRETP, the value in its timestamp field is copied and saved in the timestamp field of its corresponding ACK packet.

The source CRETP will check the effectiveness of the receiving ACK. An ACK is only effective when its sequence number equals the sequence number of current data packet in source CRETP. It implies that a delayed ACK is considered to be effective and will be discarded. Negative acknowledgments are not used in CRETP.

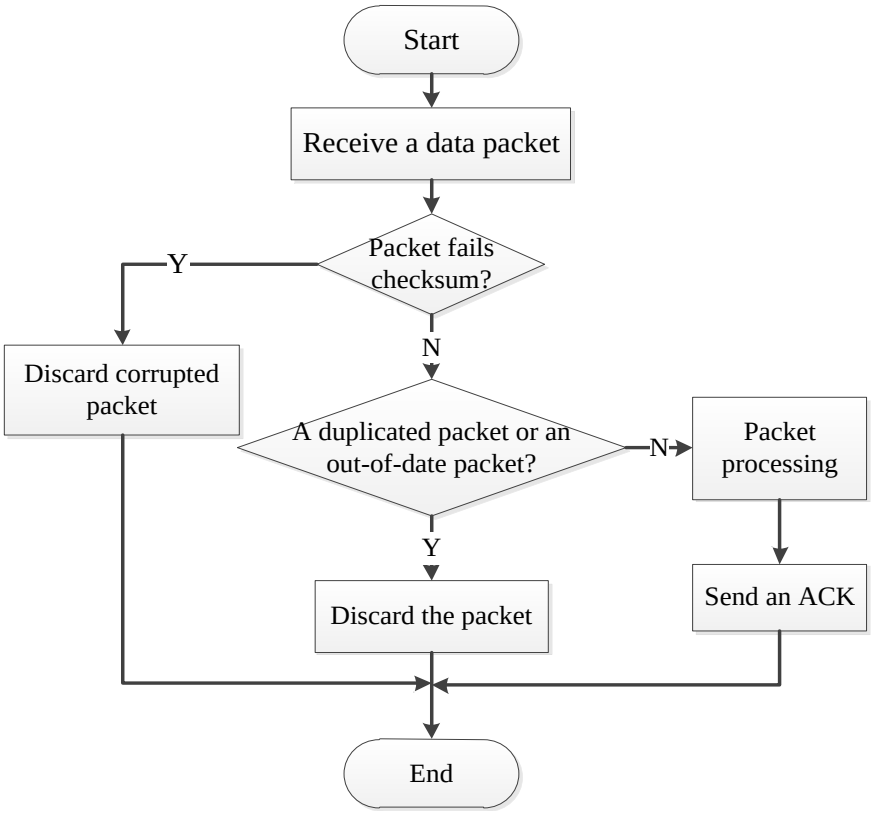

Figure 4.11: Packet effectiveness detection

4.3.3 Conditional Retransmission Mechanism

To compensate for lost or discarded packets, the source CRETP has the ability to retransmit those packets to their destinations. This conditional retransmission mechanism is the key feature of CRETP.

Since CRETP is designed for real-time data transmission in an NCS, it must achieve data timeliness while providing reliable transmission. A unique feature of real-time control systems is the predominantly periodic traffic pattern, and the traffic load for each control loop is known in advance with a fixed control frequency [Cena et al., 2008]. A control packet is effective within its control period, and it becomes useless when a new control packet becomes available at the start of a new control period. Therefore, retransmission of the current control packet has to be stopped when a new control period starts and the new control packet is ready to be sent. This is the

constraint of retransmission in CRETP.

Like other retransmission enabled protocols, CRETP employs a retransmission timer that handles the waiting time for an acknowledgment of a data packet. When the source CRETP sends a data packet, it creates a retransmission timer. Then, three situations may occur:

1. If an acknowledgment is received for this particular data packet before the timer expires, the timer is cancelled;

2. If the timer expires before the acknowledgment arrives, the data packet is retransmitted and the timer is reset;

3. If CRETP is called by the application layer to send new data, the timer is cancelled and CRETP starts transmission of the new data.

However, there is a problem: how to choose the time-out value of a retransmission timer? CRETP is a connectionless protocol, which means that each CRETP packet is independent and may travel on different paths. Different paths have different lengths. Selecting a fixed retransmission time for all data packets in a certain communication sequence can result in serious consequences. For example, if the time period between sending a data packet and receiving the corresponding ACK is much longer than the retransmission time, it may result in retransmission of a redundant data packet. Transmitting redundant packets wastes network resources. Conversely, if the retransmission time is much longer than necessary for a short path and a retransmission is needed, it can cause the delay of data retransmission which is not in favour of delay sensitive applications. Even in the case where all sending data packets choose the same path, the retransmission time should not be constant when the network's condition is taken into account. A packet can be transmitted much faster during non-traffic periods than in a congested period.

In order to provide efficient retransmission, CRETP uses a dynamic retransmission time that may be different for each data packet and can be changed during the communication. In CRETP, the retransmission time is made dynamic by basing it on

the average value of the round-trip time (RTT_e). Formula 4.1 shows how the RTT_e is applied in the calculation of the retransmission time.

$$Retransmission\ time\ =\ \beta\ \times\ RTT_e \tag{4.1}$$

In the above equation, factor β must be larger than 1. β's value is usually set based on the network's condition or past experience. When β is small, e.g., β is just a little over 1; CRETP can know about packet loss as soon as possible and starts retransmission immediately. However, it may cause a relatively large number of retransmissions and burden the network. If β has a large value, CRETP has to experience a long waiting time. Due to CRETP's constraint of retransmission, a lost packet can not be retransmitted if new data becomes available before the retransmission time-out. Therefore, a long waiting time may result in unreliable data transmission. The most common value of β is 2, and we recommend that the value of β is in the range from 1.5 to 2.

Calculation of RTT_e should be dynamic as well. CRETP measures the time between the sending of the first data packet and the receipt of its acknowledgment. This time is the first round-trip time of a communication between the source and destination. The value of RTT_e used in the calculation of the retransmission time of the following data packet is then updated according to Formula 4.2.

$$RTT_e = \alpha\ \times\ RTT_p\ +\ (1-\alpha)\ \times\ RTT_c \tag{4.2}$$

RTT_p is the previous RTT_e and RTT_c is the round-trip time of the latest data packet. The source CRETP calculates RTT_c as soon as it receives an effective ACK for the current data packet. Constant α is a smoothing factor between 0 and 1. It affects the reaction of RTT_e to changes of the network's condition. When α is chosen to be close to 0, like 0.001, RTT_c plays the dominant role in the calculation of RTT_e. This implies that the current network condition may significantly affect the average round-trip time of the communication, which is not preferred. For example, the present RTT_c might be

very large due to a transient error in the network channel. With a small α, the updated RTT_e is exaggerated, which reduces the efficiency of retransmission if network channel recovers very soon. Therefore, our recommended value of α is in the range from 0.8 to 0.9.

In the estimation of the present RTT_c, a problem would occur in the situation described as follows. Suppose that the data packet is not acknowledged before the expiry of the retransmission timer, and it is therefore resent. When the source CRETP node receives an ACK for this data packet, it does not know if the acknowledgment is for the original data or for the retransmitted one. If the original data was lost or discarded and the ACK is for the retransmitted one, the value of RTT_c should be calculated based on the departure time of the resent data packet. Conversely, if the original data was accepted by the receiver and the ACK is for the original one, the value of RTT_c should be calculated based on the departure time of the original data packet. This dilemma was solved by Karn [Forouzan and Fegan, 2003]. Under Karn's algorithm, the RTT of a retransmitted packet is simply not measured. The value of RTT_e should not be updated until the CRETP node receives an acknowledgment for the current data packet without any retransmission. This completely eliminates the problem of acknowledgment ambiguity.

However, when Karn's algorithm is used, increased delays due to retransmissions are not detected, which cannot be used to update the average round-trip time. Hence, further modification is needed for the calculation of retransmission time. A timer back-off scheme is applied along with Karn's algorithm in CRETP to prevent retransmissions from being sent too frequently. When a packet retransmission occurs, the retransmission timer is rescheduled to a back-off value rather than the same value it was set for the initial retransmission. The back-off value is set to be twice as much as the previous retransmission time when a "back off" happens for the first time. There exists the probability that retransmission for a particular data packet happens several times. In this circumstance, the back-off value continues to grow every time a retransmission happens using a multiplier (typically 2) until a retransmission is successful or it reaches

its maximum. In our simulations, the maximum of a back-off value is 6 times that of the present RTT_e by constraining the maximum number of retransmission to be 3 for each packet.

Figure 4.12 shows the packet sending procedure at the source CRETP. The dashed block represents a retransmission loop. CRETP will terminate the on-going process in a retransmission loop if it receives new directions from the application layer. This implies that retransmission for the current packet is only allowed before new data comes.

4.4 Protocol Implementation in NS-2

Experimentation and simulation are two different methodologies suitable for protocol evaluation. However, a large-scale experiment, e.g., tens of hosts communicating with each other in an area of 250 square meters, can be labour intensive, costly and logistically difficult.

Simulation is a cheaper and easier-to-implement assessment method for protocol design and evaluation. A well designed simulation tool can provide comprehensive and precise simulations. NS-2 is a well-known and widely-used discrete event-driven simulator with a very rich library of network and protocol objects [Groups, 2007]. It provides a split-level programming model in which the detailed definition and operation of protocols is done in the C++ programming language while simulation setup, such as the network topology and the specific protocol that we wish to simulate, is done in an object oriented scripting language (OTcl) [Altman and Jimenez, 2003].

There are many advantages when NS-2 is used as the simulation tool for CRETP. Due to NS-2's class hierarchy, it is easy to implement the interface that lets upper layer protocols call CRETP. The protocol itself is implemented as a protocol agent in NS-2 using C++, and testing this new protocol under appropriate network conditions is achieved by developing Tcl scripts. The simplicity of the Tcl scripting language makes it easy to generate different simulation scenarios, while C++'s richness makes

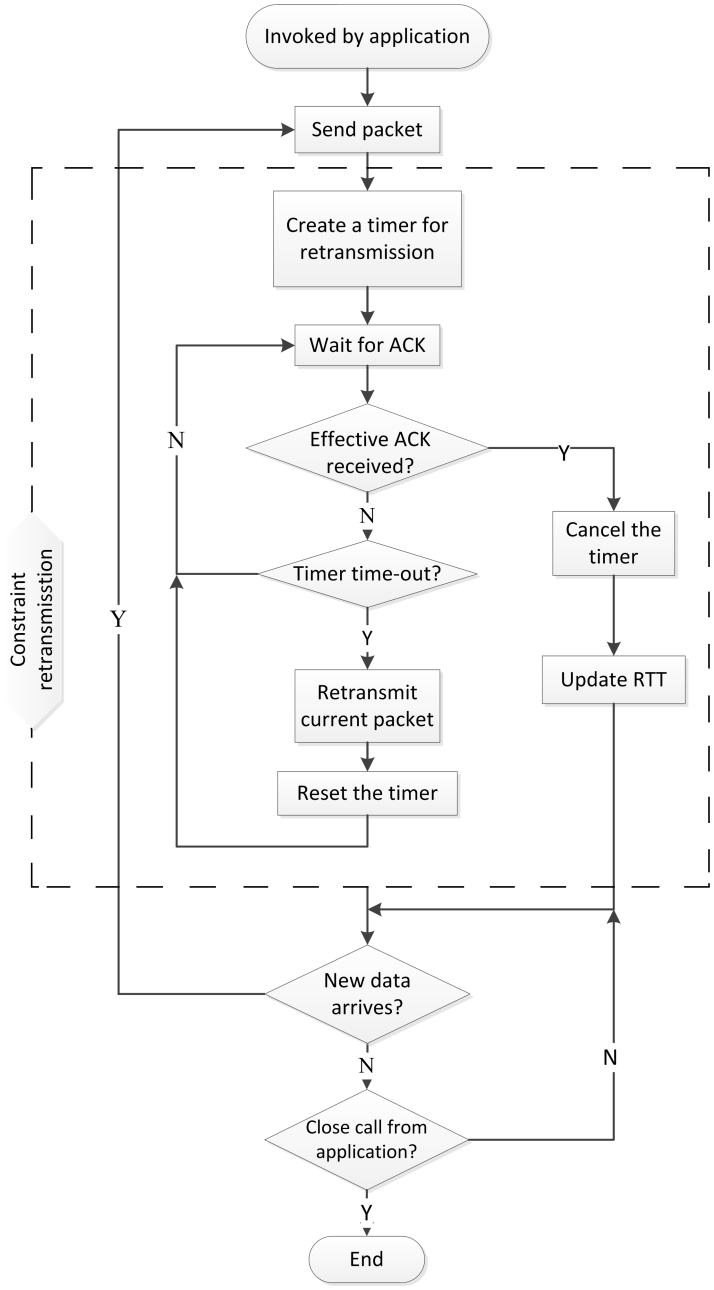

Figure 4.12: CRETP's packet sending procedure

CRETP efficient to run.

Figures 4.13 to 4.18 present detailed sub algorithms for CRETP's main operations when it is implemented in NS-2. CRETP's working mode depends on how it is invoked by an application. If it is an active open call from an application, CRETP works as a

source sending data. Conversely, CRETP works as a destination receiving data when it gets a passive open call. The following two sections elaborate the implementation algorithms of CRETP's operations in NS-2 from the view of a source CRETP and the view of a destination CRETP respectively.

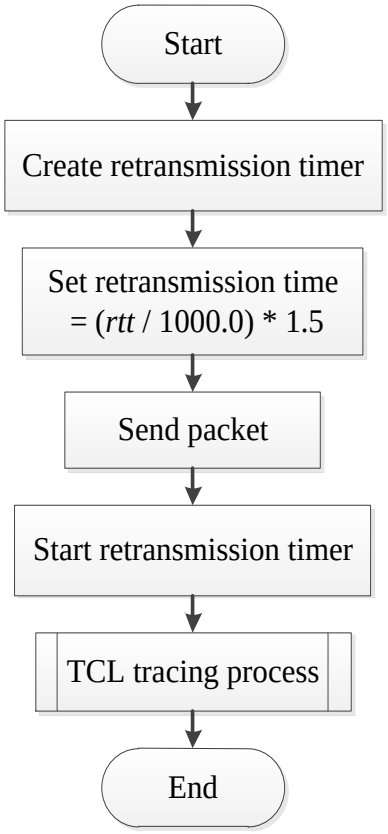

Figure 4.13: Data packet sending process in a source CRETP

4.4.1 Main Operations in Source Mode

CRETP is invoked and gets data from applications when the *sendmsg*() function is called. When CRETP works as the source in a communication, the packet type is initialized as a CRETP data packet. Several parameters must be initialized before the start of communication. For instance, the sequence number is initialized as -1. When selecting the initial value for variable *rtt* which represents the average RTT,

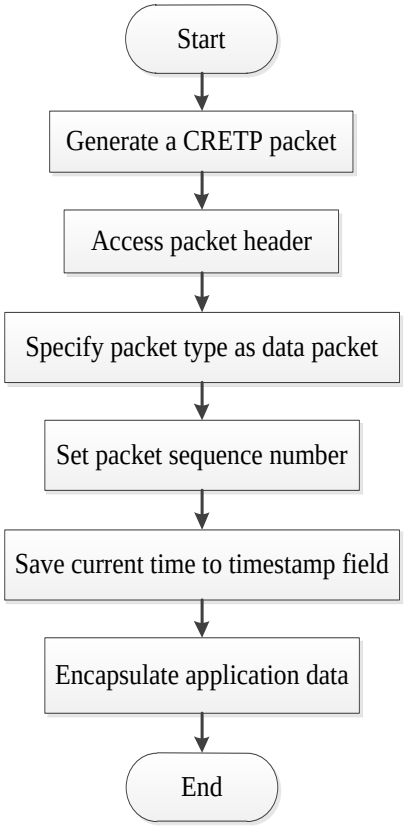

Figure 4.14: Data packet generation in a source CRETP

some elements, such as the network scale's and condition, can be taken into account to improve CRETP's efficiency. During our simulations, *rtt* was usually initialized to 10 milliseconds, so the initial retransmission time was 20 milliseconds.

After initialization, CRETP starts the communication with the addressed destination. The main operations of a source CRETP node include sending the data packet, retransmitting the current data packet if necessary, and receiving and processing acknowledgements from the destination.

Figure 4.13 shows that algorithm for the transfer of a new packet, while Figure 4.15 describes CRETP's retransmission process. Note that the retransmission process may be terminated if an effective ACK is received or a new direction from the application arrives. The sub process of data packet generation in Figure 4.13 is elaborated in Figure 4.14. CRETP has the ability to divide application data into small parts if the

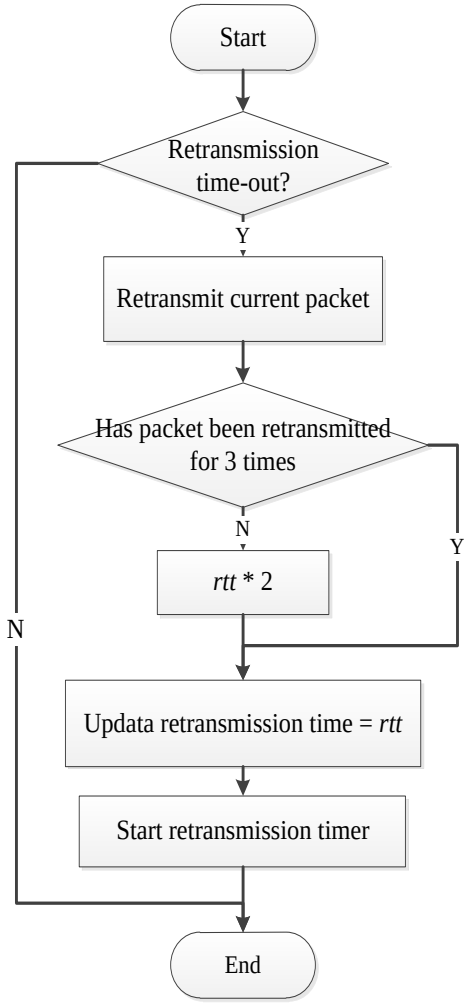

Figure 4.15: Retransmission process in a source CRETP

data is too big for a CRETP packet. However, since a real-time control packet is typically much smaller than a CRETP packet, Figure 4.14 only shows the case where the entire application packet is sent within a single CRETP packet. Besides sending data packets, the source CRETP receives ACKs from the destination. Figure 4.16 shows the process for receiving and handling ACKs in the source CRETP node.

4.4.2 Main Operations in Destination Mode

The *recv*() function in CRETP will be called by an application to start receiving CRETP data packets from a sender. When CRETP acts as the destination in a communication,

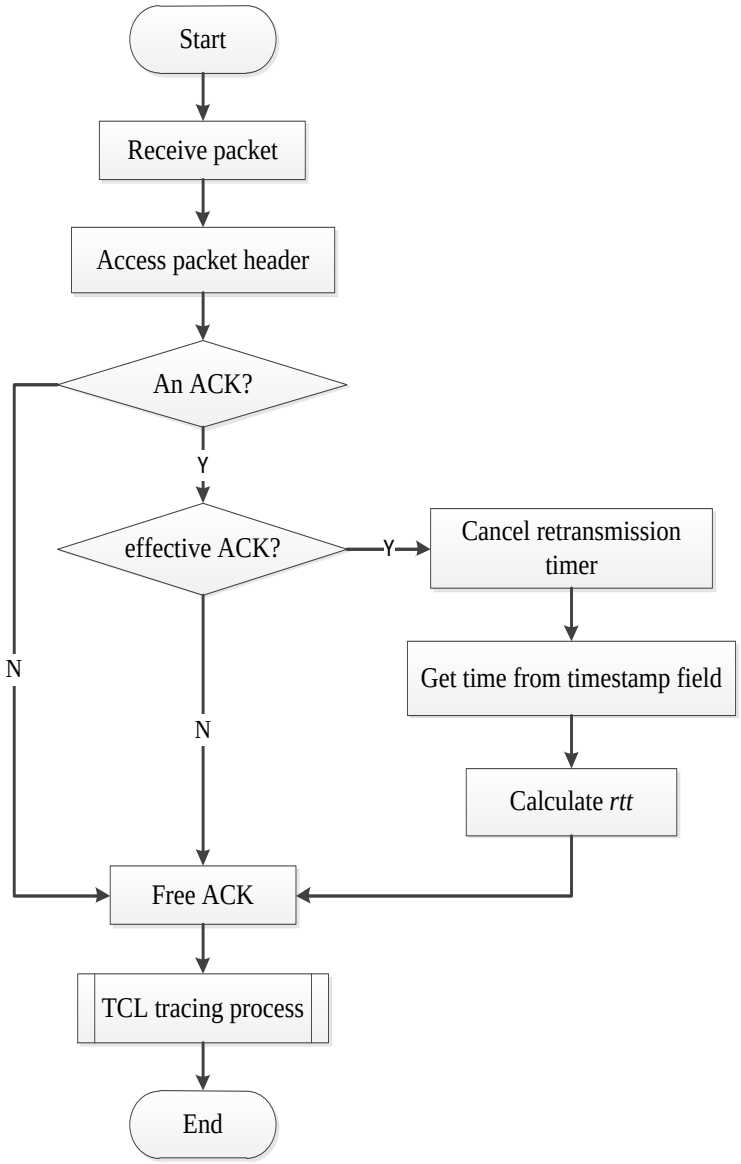

Figure 4.16: ACK receiving and processing in a source CRETP

the returned packet type is initialized as a CRETP ACK packet. The initial value for *seq* (the variable for sequence number checking) is 0.

The main behaviours of a destination CRETP includes receiving data packets from the source, checking data effectiveness, and acknowledging effective packets. Figure 4.17 shows CRETP's packet receiving process. Data effectiveness detection is included in this flow chart. It can be observed that CRETP only recognizes the effective

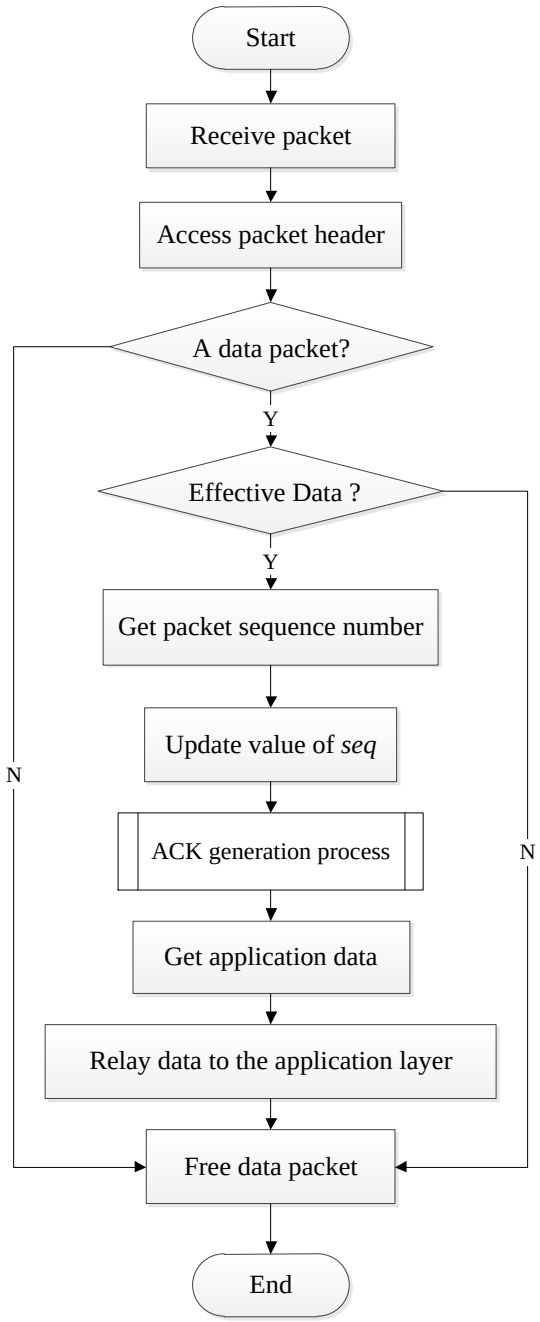

Figure 4.17: Data packet receiving process in a destination CRETP

data. It ignores the difference between duplicated data and an out-of-date packet, and simply discards both of them. This is implemented by comparing the sequence number of the current data packet with the value of *seq*. Variable *seq* records the next sequence

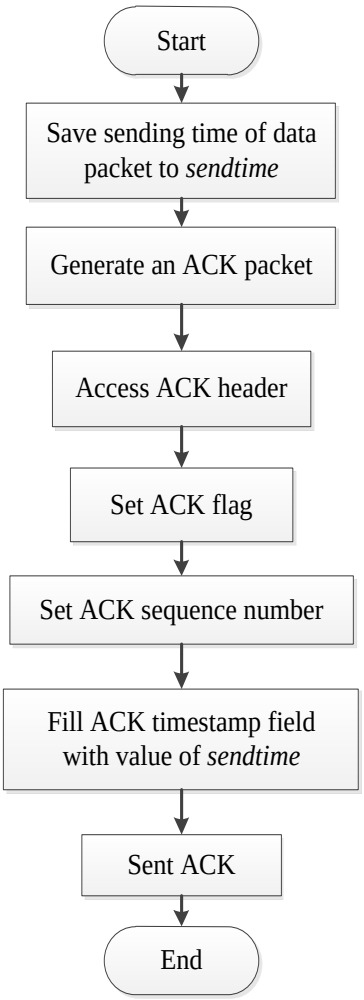

Figure 4.18: ACK generation in a destination CRETP

number that is expected by a CRETP node. A packet is considered effective only when its sequence number is not less than *seq*.

The receiving action for the first effective data packet is a bit different from the receiving process for others. When the first uncorrupted data packet arrives, CRETP does not check its sequence number. Instead, CRETP saves its sequence number in *seq* and increases *seq* by 1.

The process of ACK generation is illustrated in Figure 4.18. In our simulation, the size of a CRETP ACK packet is set to be 64 bytes.

The source code of CRETP with explicit comments is given in Appendix B. Please

refer to them for detailed CRETP implementation in NS-2.

4.5 Chapter Summary

A transport layer protocol compensating for data losses with data timeliness in mind was presented in this chapter. This conditional retransmission enabled protocol, CRETP, is designed for real-time communications in NCSs where every data packet has a deadline. A data packet becomes ineffective when its deadline expires or newer data becomes available. CRETP is able to check the effectiveness of real-time data and will not retransmit an invalid data packet. It guarantees data timeliness as well as saving network resources. In order to improve the efficiency of retransmission, CRETP also employs a dynamic calculation of retransmission time by updating the average measured round trip time (RTT_e) every time when a retransmission occurs.

Performance evaluation of CRETP was done through extensive simulations and is presented in the following chapter. Comparative studies of simulation results between CRETP, UDP and TCP are also performed to show CRETP's advantages when used for wireless real-time communications in an NCS.

Chapter 5

CRETP Performance Evaluation in NS-2

In order to provide a comprehensive performance evaluation for CRETP, extensive simulations were conducted. Since CRETP was designed as a network protocol for real-time control systems, delay performance and transmission reliability are two essential criteria. An NCS requires small and deterministic delays. Also there is no doubt that the more reliable the data transmission is, the better is the network performance of an NCS. As mentioned in Chapter 1, the overall performance of a real-time wireless NCS is not evaluated because the control performance, which is another aspect of NCS performance, is beyond our research scope.

Evaluation of CRETP was conducted through comparative studies for the behaviour of all three protocols, UDP, TCP and CRETP. This is because we want to see:

- if CRETP can perform better than UDP and TCP in real-time control systems in terms of reliability; and
- if CRETP can keep its delay performance at an acceptable level for real-time applications. For example, are the delays introduced by CRETP smaller than

TCP's and are close to UDP's ?

Only if the above questions get positive answers, can we claim that CRETP has the ability to greatly improve transmission reliability and keep data timeliness for real-time applications in NCSs. It will also mean that CRETP distinguishes itself among the three protocols by showing the best overall performance, which would make it more suitable than UDP and TCP for real-time control systems.

Our simulations were conducted for three case studies. The performance of a protocol depends on many elements in a wireless network. Traffic load and wireless channel condition are two major factors that influence the overall performance of a protocol. More specifically, traffic load varies with the change in the number of communicating nodes and the control period. Therefore, in our simulations, variations in channel conditions, the number of communicating sensors and the control period, were investigated as the three primary parameters to define three different case studies. Usually, the control period is pre-determined in control algorithm design and it is used as a constraint in this work.

For all the scenarios in Case One, the number of communicating devices and the control period were the same, leaving the channel's condition the only variable parameter. In the simulation scenarios of Case Two, the wireless channel condition was kept fixed, while the control period was set with different values in different scenarios. Case Three employed the number of sensors as the new parameter which affects traffic load significantly.

5.1 Network Specification

The aim of simulations was to test CRETP's performance for real-time applications in wireless NCS. Therefore, we simulated communications between sensors and the controller in control loops. Compared with other wireless network applications, the number of sensors, controllers, actuators, and other devices interconnected within a physical subtask in a wireless NCS is typically low, e.g., a few tens or less [Cena et al.,

2008]. The maximum number of communicating devices in our simulation scenarios is 11.

All together twenty-six scenarios were simulated. For the consistency of the comparative studies, all scenarios have to share some basic settings. Table 5.1 lists basic configurations of the wireless network model that was used in all scenarios. Besides using the same basic wireless model, all the simulated scenarios have some common presumptions specified as follows:

- All sensors and the controller are fixed in the network field with sensors distributed in a circle around the controller, which implies that distances between the controller and sensors are the same and constant as the circle's radius is always 50 metres. Figure 5.1 demonstrates the network layout used in the simulations.
- The simulation time for each scenario is 16.0 seconds.
- All sensors start their communications with the controller at 1.0s and stop at 16.0s.
- With respect to an NCS's predominantly periodic traffic pattern, all sensors generate data packets periodically. Sensors are designed to generate a control data packet at the beginning of each control period.
- The traffic type on the application layer in sensors is Constant Bit Rate (CBR).
- As application data packets in an NCS are usually fixed in size and are typically short, e.g., a few hundreds or even tens of bytes, the application data in the simulations has a fixed size of 200 bytes.

The aim of simulations was to test the performance of our newly designed transport layer protocol. Therefore, communication between sensors and the controller was simulated other than the whole control loop.

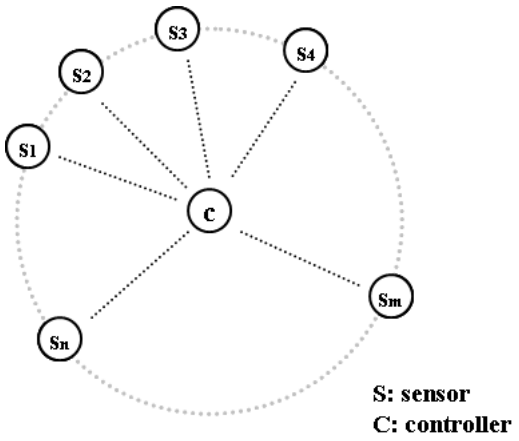

Figure 5.1: Network topology in simulations

Table 5.1: Wireless model used in simulations

Network standard	IEEE 802.11b
Radio channel data rate	1.0 Mbps
Network area	250m * 300m square
Radio-propagation model	Two-ray ground
Routing protocol	Dynamic Source Routing
Wireless interface (MAC) buffer type	Drop-tail priority queue
AntennaD	OmniAntenna

5.2 Performance Metrics

Comparative studies of dynamic behaviour among these three protocols were conducted for four performance metrics: end-to-end delay, the data loss ratio, the percentage of effective data received at the controller and the occurrence of consecutive dropouts. All these metrics are chosen from the networking aspect as we aim to investigate protocols' influences on network performance of a wireless NCS.

Each packet generated from a sensor has a non-zero travel time if it is sent to its destination through a network. The travel time begins the moment when the packet is delivered at the source sensor from the application layer to the transport layer. It ends when an effective copy of the packet is relayed by the transport layer protocol at the destination to the application layer. This travel time is known as a packet's end-to-

end delay. In our protocol evaluations, both the individual end-to-end delay for each received data packet and the average delay for all data packets were used as metrics for protocol performance since one of an NCS's main requirements is a small and deterministic delay. The average delay is obtained by summing up the individual data packet delays in the network, and dividing the sum by the total number of received data packets.

Packet dropouts happen in network communications due to many reasons and can be compensated for by employing proper network protocols. When the network's condition gets worse, packet dropouts may occur often. The number of dropped packets is a criterion of transmission reliability and represents a protocol's ability to provide reliable data transfer.

In real-time control systems, data timeliness is essential as a data packet is only useful during its control period and becomes ineffective when a new data packet for the next control period arrives. Therefore, the data loss ratio is not the most accurate metric for transmission reliability. Instead, the percentage of effective data packets received in the network has to be measured as it is the most appropriate metric for demonstrating reliability of a protocol in real-time communications.

System performance is also sensitive to the pattern of packet dropout. For the same percentage of effective data, if dropouts happen consecutively, the control performance may degrade to be an unacceptable level due to the continuous misses of control law updates. It implies that the occurrence of consecutive data dropouts has to be minimized to enhance system stabilization. Therefore, the numbers of occurrence of two or more consecutive dropouts in each simulation were recorded as the fourth metric examining protocol performance.

5.3 Simulation Case One

5.3.1 Simulation Scenario Specifications

A feature of current wireless networks is that the radio signal is always subject to physical conditions like interference and signal attenuation, which makes it difficult for a wireless network to meet the communication requirements in real-time control applications [Cena et al., 2007]. As a major factor in network protocol performance, the channel's condition was employed as the only variable parameter to draw distinctions between simulation scenarios in the first case study.

Eleven different wireless channel conditions were simulated separately in scenarios one to eleven. The difference between these wireless conditions was the number of irregular run-time channel errors happening during the simulation, as shown in Table 5.2. An irregular run-time channel error means the error occurs at a random time with an independent duration. Due to wireless channel errors, packet transmissions can be delayed and even packet dropouts can occur. Generally speaking, the greater the number of errors, the worse the channel condition. The network condition in Scenario 11 was the worst among all scenarios.

Table 5.2: Number of channel errors in each scenario

Scenario	1	2	3	4	5	6	7	8	9	10	11
Number of channel errors	4	7	9	10	12	15	18	20	22	25	28

As mentioned in the previous section, the network traffic load was constant in all scenarios in Case One. The number of sensors was 5 while the control period was 50 milliseconds. Given the number of sensors and the control period, we can figure out that there are 1500 different data packets to be sent from the sensors during a 16.0-second simulation (traffic starts at 1.0 second). In the case of retransmission, more data packets are sent, of course.

All eleven scenarios were simulated three times. For the first time, the transport layer protocol used was the TCP protocol. Then, UDP was substituted for TCP when all scenarios were tested for the second time. Finally, CRETP acted as the transport protocol in all communications.

5.3.2 Simulation Results

Table 5.3 records the average end-to-end delays for all successfully received data packets in the different scenarios when using different transport layer protocols. These results are also shown graphically in Figure 5.2. The numbers of dropped data packets in communications are listed in Table 5.4, while Table 5.5 records times of occurrences of consecutive dropouts in each simulation scenarios. Table 5.6 summarises the percentage of effective data packets received at the controller in different scenarios with different protocols. The results in Table 5.6 are graphed in Figure 5.3. Table 5.7 shows numbers of two or more consecutive losses of effective data packets at the controller while Table 5.8 lists maximum number of consecutive data losses in each scenario.

Table 5.3: Average end-to-end delays in Case One when using different protocols

Scenarios	Average end-to-end delay (milliseconds)		
	UDP	CRETP	TCP
1	10.061991	11.636312	12.291061
2	10.085596	12.107675	16.319366
3	10.055195	11.968628	18.127047
4	10.070746	12.216363	19.050467
5	10.057896	12.104100	20.709423
6	10.065880	12.465714	22.270711
7	10.040047	12.566186	23.382781
8	10.064531	12.719966	23.346436
9	10.081729	12.732849	24.231340
10	10.064618	13.306682	30.518884
11	10.089902	12.529813	37.585497

Besides the overall simulation results in Case One, specific illustrations of delays for each scenario were recorded. As a specific example, Figures 5.4, 5.5 and 5.6

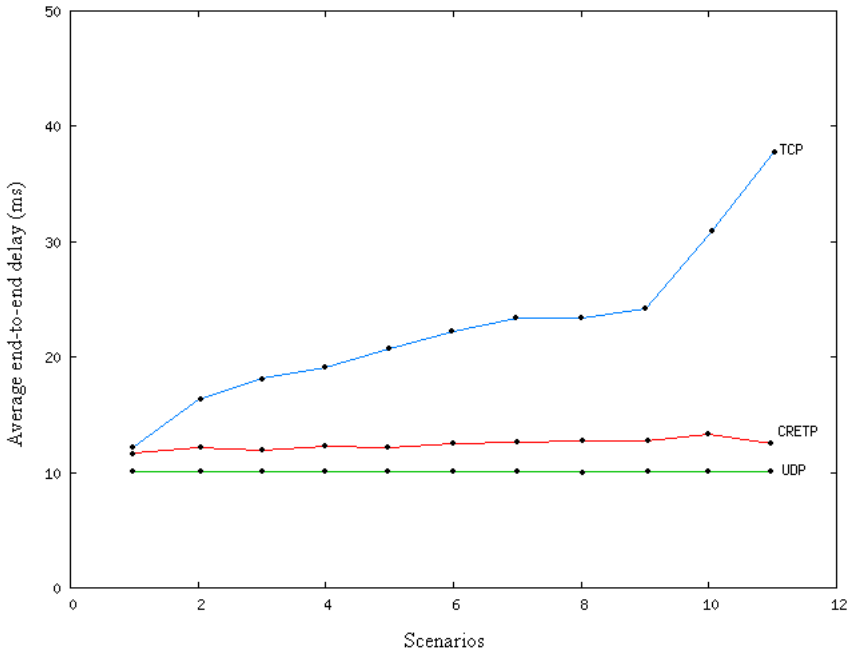

Figure 5.2: Average end-to-end delays in Case One with different protocols

Table 5.4: Number of dropped data packets in Case One when using different protocols

Scenarios	Number of dropped data packets)		
	UDP	CRETP	TCP
1	4	2	0
2	32	16	0
3	46	28	0
4	50	22	0
5	70	46	0
6	80	43	0
7	88	42	0
8	92	43	0
9	93	46	0
10	148	85	0
11	143	113	0

illustrate delays for all successfully received data packets sent by sensor 1 in Scenario 7 when using UDP, CRETP or TCP, respectively.

Table 5.5: Consecutive dropouts in Case One with different protocols

Scenarios	Number of occurrences of consecutive dropouts		
	UDP	CRETP	TCP
1	0	0	0
2	11	0	0
3	15	5	0
4	19	2	0
5	13	6	0
6	15	3	0
7	18	7	0
8	16	3	0
9	20	5	0
10	36	18	0
11	49	23	0

Table 5.6: Percentage of effective data received in Case One with different protocols

Scenarios	Percentage of effective data packet)		
	UDP	CRETP	TCP
1	99.733%	99.867%	99.8%
2	97.867%	98.933%	97.933%
3	96.933%	98.133%	96.933%
4	96.667%	98.533%	96.667%
5	95.467%	96.933%	95.333%
6	94.667%	97.133%	94.6%
7	94.133%	97.2%	94%
8	93.867%	97.133%	93.733%
9	93.8%	96.933%	93.467%
10	90.133%	94.333%	90.2%
11	90.267%	92.467%	90.933%

5.3.3 Comparative evaluations

When we look at the average end-to-end delay (Table 5.3 and Figure 5.2), everything seems in favour of a UDP connection: It yields the smallest average delay among the protocols. CRETP is in second place providing average delays with slightly higher values, while TCP acts worst.

Table 5.7: Consecutive losses of effective data in Case One with different protocols

Scenarios	Number of occurrences of consecutive losses of effective data		
	UDP	CRETP	TCP
1	0	0	0
2	11	0	11
3	15	5	14
4	19	2	18
5	13	6	14
6	15	3	17
7	18	7	20
8	16	3	17
9	20	5	22
10	36	18	37
11	49	23	46

Table 5.8: Maximum number of consecutive data losses in Case One with different protocols

Scenarios	Maximum number of consecutive data losses		
	UDP	CRETP	TCP
1	0	0	0
2	3	0	3
3	3	3	3
4	3	3	4
5	3	3	4
6	3	3	3
7	3	3	4
8	3	3	3
9	3	3	3
10	3	3	3
11	4	3	4

When UDP was used in Case One, the average delays under this amount of traffic load were limited within 10.09 milliseconds, regardless of the channel's condition. As for CRETP, the average delays are just slightly longer than the average delays provided by UDP, with an upper bound of 13.31 milliseconds. It also can be seen

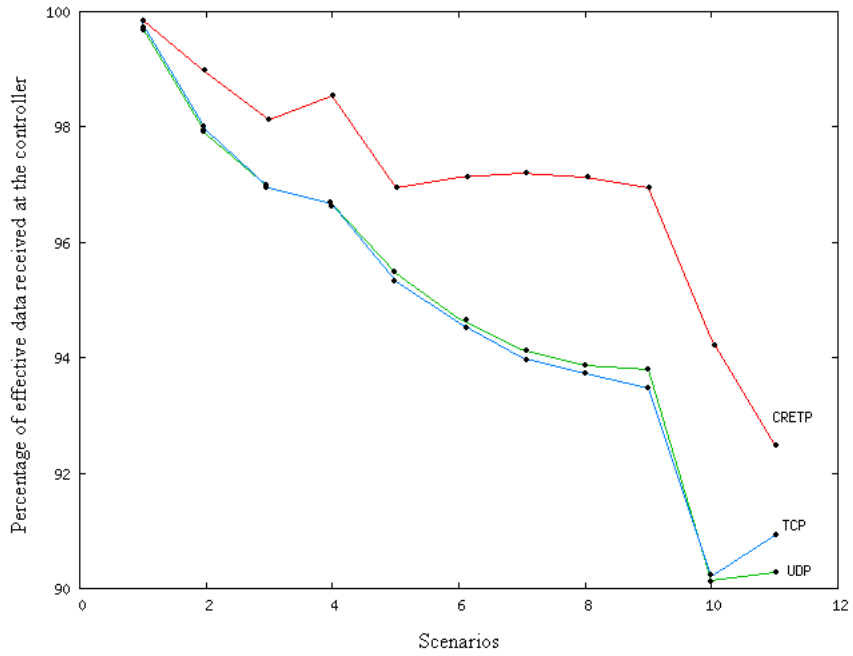

Figure 5.3: Percentage of effective data received in Case One with different protocols

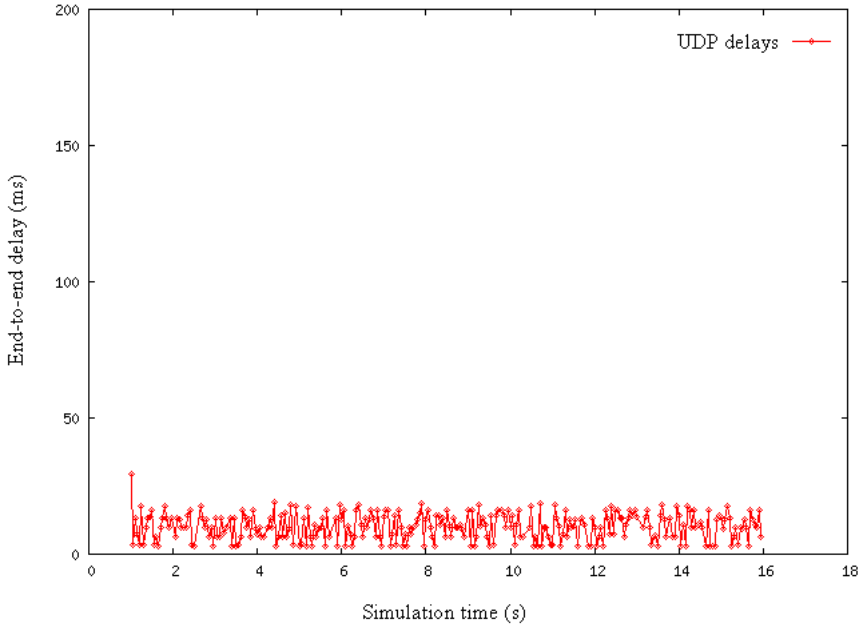

Figure 5.4: End-to-end delays for sensor 1 in Scenario 7 when using UDP

from Figure 5.2, for UDP or CRETP, that the average delays are almost constant as the network's condition gets worse. Small and deterministic delays are desirable for

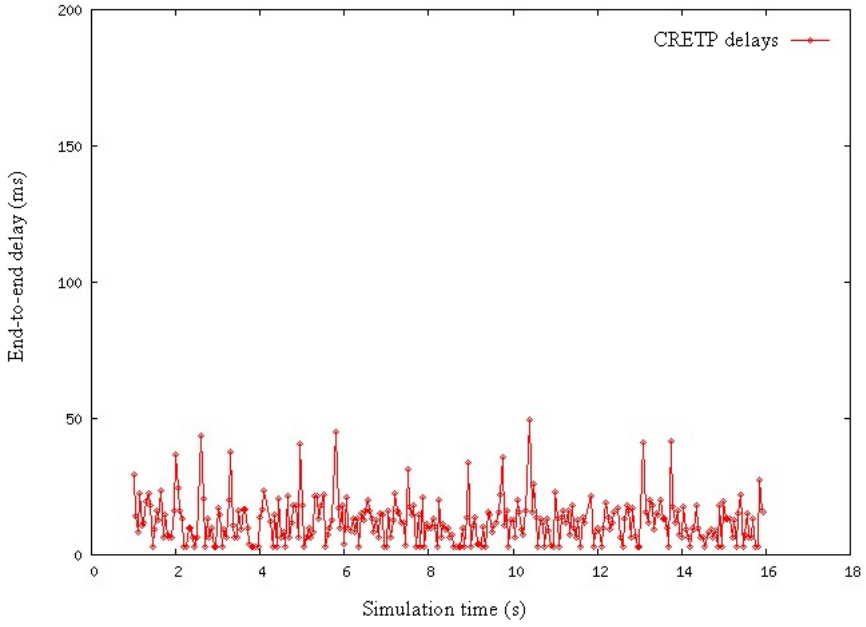

Figure 5.5: End-to-end delays for sensor 1 in Scenario 7 when using CRETP

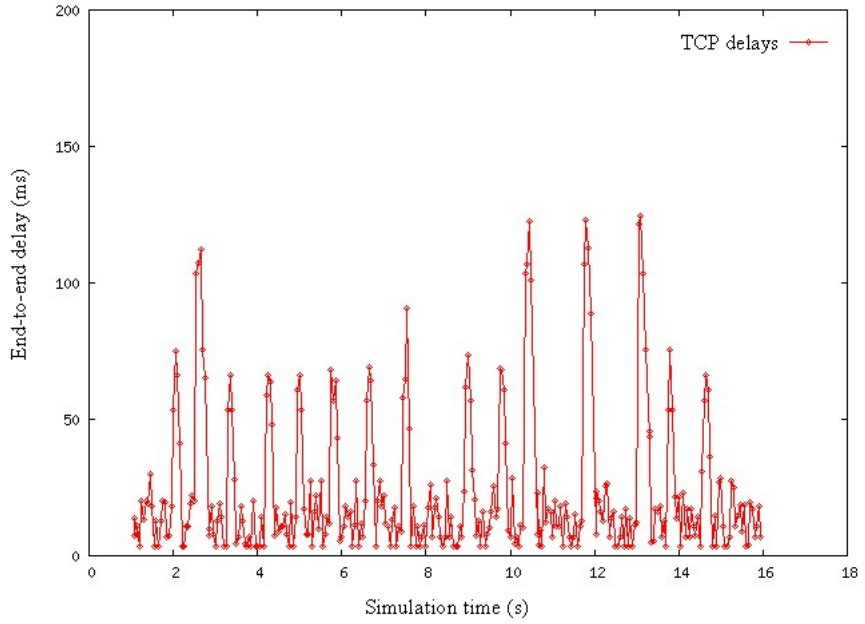

Figure 5.6: End-to-end delays for sensor 1 in Scenario 7 when using TCP

real-time applications. According to these simulation results, both UDP and CRETP are suitable for real-time communication in terms of end-to-end delay.

However, when TCP is employed, the average delay increases almost linearly

with the number of channel errors. The 37.585497 milliseconds ceiling is reached in Scenario 11. Almost 75.2% of a control period is spent on the successful transfer of a single data packet, leaving the time available for other operations very limited. Another unfavourable characteristic of these relatively long delays is the indeterminism. All these results suggest that TCP is not a suitable network protocol for real-time control systems.

Figures 5.4 to 5.6 give close-ups of the delay performances for the three protocols in Scenario 7. Each dot in the figures represents the arrival of a data packet at the controller. It can be observed that almost every effective data packet is received within 30% of a control period when UDP is the transport protocol. As for CRETP, most packets arrive at the controller before half of their control period elapses. The rest of the packets reached their destination later but still within their control period (50 milliseconds). As for TCP, many packets spent more than one control period travelling from the sensor to the controller. Some of them even took more than twice a control period to finish their transmission. All these packets are therefore ineffective.

The reason why TCP cannot perform well is because of its endless retransmission scheme. A packet will be retransmitted until its corresponding ACK arrives at the sender. Repeated retransmission leads to long end-to-end delays. In the case of a channel error, retransmission occurs more frequently than in a normal channel condition. The delay difference between a packet successfully transferred in one step and a packet being retransmitted several times can be very obvious. Therefore, both the average end-to-end delay and jitter for TCP produced comparatively large values.

Although CRETP has the ability to retransmit a packet if needed, the retransmission is conditional. If a new application data packet is passed to CRETP when a new control period starts, CRETP will stop retransmitting the old data packet. This implies that each data packet will be retransmitted only during its control period. For real-time applications, expired data is useless and will be dropped by the receiving end even if it is received. In this case, a high delay could be translated to a high packet loss rate. CRETP's restriction on the number of retransmissions prevents the packet end-to-end

delay from becoming excessive and keeps delays within a certain range.

UDP distinguishes itself in terms of its transmission latency. However, when it comes to transmission reliability, UDP's performance degrades. The UDP column in Table 5.4 shows that packet loss starts to grow without bound as the network's condition gets worse. CRETP also introduces dropouts due to its conditional retransmission scheme. The dropped data packets counted for CRETP include packets which failed to reach the receiver and packets tested as ineffective. Dropout numbers were smaller when compared with those provided by UDP, however.

CRETP did not only introduce less packet losses than UDP, it also reduced the frequency of occurrence of consecutive dropouts. Table 5.5 shows that in most cases, consecutive dropouts only happened several times when CRETP is used. However, in the case of UDP, consecutive dropouts occurred more than ten times in all simulations except in the first scenario when channel's condition was good.

No dropouts occur when TCP is the transport layer protocol. However, when data effectiveness is taken into account, TCP does not provide the most reliable transmission for real-time applications. Table 5.6 and Figure 5.3 demonstrate percentage values of the ratio of effective data packets received at the controller to the total number of sent data packets. Although TCP's retransmission ensured that all data packets were received at their destinations, it cannot guarantee the effectiveness of received data. No matter what transport protocol is used, the ratio of effective data decreases as the network's condition deteriorates. CRETP's conditional retransmission scheme enhances communication reliability while keeping data timeliness in mind, which means it always, had the highest value of percentage of effective data packets among the three protocols and guaranteed a percentage of 92.467 in the worst case scenario.

In Table 5.7, it also can be found that CRETP's conditional retransmission scheme protects data effectiveness and guaranteed no consecutive data losses when both UDP and TCP started to fail in continuously reliable transmission of data packets. However, if channel condition gets worse, consecutive data losses occur no matter which protocol is in use. In the cases of consecutive dropouts, CRETP provides the smallest number of

occurrence of consecutive effective data losses at the controller among three protocols. The maximum number of consecutive data losses is 3 when CRETP is employed. Consecutive losses of 4 effective data packet can be found in simulations with UDP/TCP acting as the transport protocol. The more consecutive data loss, the less reliable is the transmission. Records in Tables 5.6 to 5.8 evidenced that CRETP is in lead among three protocols in terms of transmission reliability.

Overall, the simulation results in Case One showed that TCP's delay performance continuously gets worse when the channel state deteriorates. Although its endless retransmission algorithm compensates for all dropouts, it fails to satisfy data timeliness, which makes TCP produce even smaller rates of effective data than UDP does in some scenarios. TCP's performance exposes its inappropriateness as a network protocol for real-time control systems.

UDP provides the smallest average end-to-end delay in all scenarios while CRETP introduces slightly larger delays due to its retransmission algorithm. For both UDP and CRETP, the transmission latency is small and relatively deterministic.

However, in terms of transmission reliability, UDP has no transmission error control which makes it perform the worst among the three protocols in some cases. CRETP produces the most reliable communication as the number of dropped packets is less than half of UDP's in most scenarios and occurrence of consecutive dropout is also much less frequent. CRETP's conditional retransmission significantly improves communication reliability while keeping delays small and relatively deterministic.

Comparative study of the performance of these protocols demonstrates that CRETP best satisfies the requirements of data timeliness as well as transmission reliability for real-time applications in an NCS with wireless networks that are vulnerable to errors.

5.4 Simulation Case Two

5.4.1 Simulation Scenario Specifications

In the previous section, the performance of UDP, CRETP and TCP in different wireless channel conditions was analysed. As another major factor influencing protocol performance, traffic load is employed as the parameter in case studies two and three. The wireless channel condition is the same in all scenarios.

As mentioned before, the overall traffic load in a network used in an NCS is defined by many elements. The control period and the number of communicating nodes are two major factors. It is obvious that the overall traffic load will increase when the number of communicating nodes in the network grows. And with a decrease of the control period, more data packets will be sent to the controller within the same time frame. Although both of these factors can affect traffic load, they may have different effects on the performance of CRETP. At the beginning of each control period, a new application data packet is generated and is about to be sent. The arrival of this new data forces CRETP to give up retransmission of the previous data packet. Theoretically, CRETP can retransmit an old packet more times in the case of a longer control period. Therefore, decreasing the control period not only produces less traffic but also gives CRETP more opportunities to conduct retransmissions. We therefore decided to use the control period and the number of sensors as parameters in case studies two and three respectively.

In Case Two, the control period was used as the only parameter for all simulation scenarios while the number of sending nodes (sensors) was set to be a constant value of 5. The only difference between the six simulation scenarios was the length of the control period. The longer the control period is, the smaller is a sensor's data rate as the size of each data packet was a fixed value of 200 bytes for all simulations. Given the number of sending sensors and the control period, we can figure out the number of different data packets that should be sent in each scenario. Table 5.9 lists the control period and the number of data packets sent by sensors, excluding retransmitted ones,

in each scenario in Case Two. It is obvious that the traffic load is on the rise from Scenario 1 to Scenario 6.

Table 5.9: Control period in each scenario

Scenario	1	2	3	4	5	6
Control period (milliseconds)	90	80	70	60	50	40
Number of data packets to be sent	835	940	1075	1250	1500	1880

All scenarios in Case Two were simulated three times. For the first time, the transport layer protocol used was the TCP protocol. Then, UDP was substituted for TCP when all scenarios were tested for the second time. Finally, CRETP acted as the transport protocol in all communications.

5.4.2 Simulation Results

Table 5.10 records average end-to-end delays for all successfully received data packets in different scenarios when using different transport layer protocols. These results are also shown graphically in Figure 5.7. Numbers of dropped data packets in communications are listed in Table 5.11, while Table 5.12 records times of occurrences of consecutive dropouts in each simulation scenarios. Table 5.13 summarises the percentage of effective data packets received at the controller in different scenarios with different protocols. The results in Table 5.13 are graphed in Figure 5.8. Table 5.14 shows numbers of two or more consecutive losses of effective data packets at the controller while Table 5.15 lists maximum number of consecutive data losses in each scenario.

Besides the overall simulation results, specific illustrations of delays for each scenario were recorded. The delay performance for sensor 1 in Scenario 3 (control period is 70 milliseconds) is randomly chosen as an example and displayed in Figures 5.9, 5.10 and 5.11. These three graphs illustrate delays of effective data packets received at the controller when using UDP, CRETP or TCP as the transport protocol.

Table 5.10: Average end-to-end delays in Case Two when using different protocols

Scenarios	Control period (milliseconds)	Average end-to-end delay (milliseconds)		
		UDP	CRETP	TCP
1	90	10.037630	15.033824	25.238436
2	80	10.131140	13.494351	24.022764
3	70	10.055708	12.758372	22.437320
4	60	10.065637	12.576783	23.442542
5	50	10.040047	12.566186	23.382781
6	40	10.044697	12.151133	23.189041

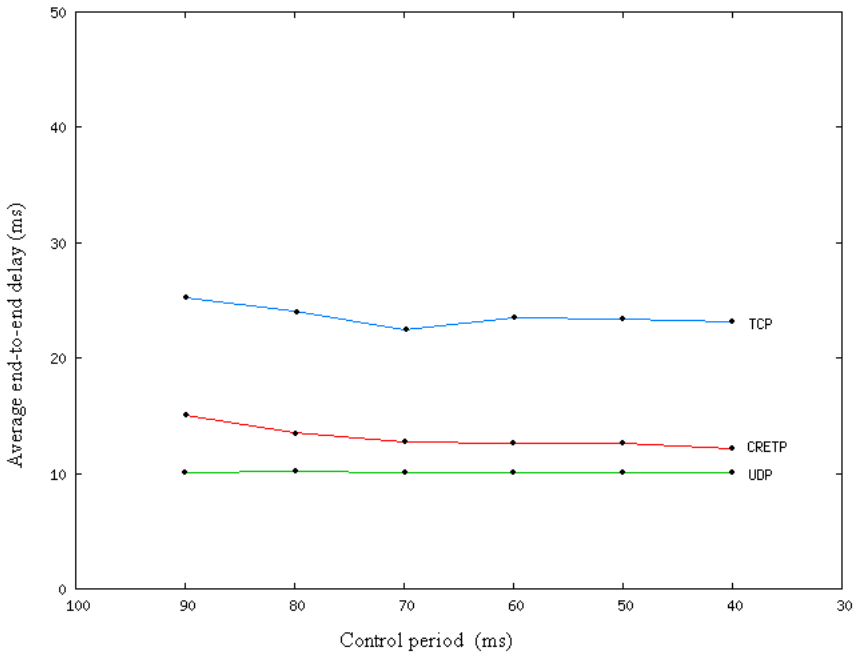

Figure 5.7: Average end-to-end delays in Case Two with different protocols

5.4.3 Comparative evaluations

When looking into delay performance, UDP still produces the smallest delays among the three protocols. As shown in Table 5.10 and Figure 5.7, CRETP also keeps the average delay small, at less than 37.5% of the smallest control period. The delays almost doubled when TCP was used.

It is easy to understand that the larger the number of data packets in a transmission,

Table 5.11: Number of dropped data packets in Case Two when using different protocols

Scenarios	Control period (milliseconds)	Number of dropped data packets		
		UDP	CRETP	TCP
1	90	53	6	0
2	80	54	11	0
3	70	53	23	0
4	60	72	35	0
5	50	88	42	0
6	40	112	82	0

Table 5.12: Consecutive dropouts in Case Two when using different protocols

Scenarios	Control period (milliseconds)	Number of occurrences of consecutive dropouts		
		UDP	CRETP	TCP
1	90	0	0	0
2	80	2	0	0
3	70	2	0	0
4	60	8	0	0
5	50	18	7	0
6	40	31	34	0

Table 5.13: Percentage of effective data received in Case Two with different protocols

Scenarios	Control period (milliseconds)	Percentage of effective data packet		
		UDP	CRETP	TCP
1	90	93.653%	99.281%	93.533%
2	80	94.255%	98.830%	93.936%
3	70	95.070%	97.860%	94.698%
4	60	94.24%	97.2%	94.32%
5	50	94.133%	97.2%	94%
6	40	93.883%	95.638%	94.043%

the busier the network. Network congestion happens if there are too many packets to be sent in the same period. Long end-to-end delays and even data losses can occur in the case of network congestion. However, even in the scenario with the smallest control

Table 5.14: Consecutive losses of effective data in Case Two with different protocols

Scenarios	Control period (milliseconds)	Number of occurrences of consecutive losses of effective data		
		UDP	CRETP	TCP
1	90	0	0	0
2	80	2	0	2
3	70	2	0	3
4	60	8	0	7
5	50	18	7	20
6	40	31	34	35

Table 5.15: Maximum number of consecutive data losses in Case Two with different protocols

Scenarios	Control period (milliseconds)	Maximum number of consecutive data losses		
		UDP	CRETP	TCP
1	90	0	0	0
2	80	3	0	3
3	70	3	0	3
4	60	3	0	3
5	50	3	3	4
6	40	4	4	4

period (40 milliseconds), the network capacity utilization is low, with a maximum value of 27.2939% for UDP, 40.6144% for CRETP and 45.3154% for TCP. This means that the traffic loads in different scenarios are still within a range in which there is no significant network congestion. Also, UDP's end-to-end delay does not ascend linearly as the number of sending packets increases but fluctuates within a range; see Table 5.10 and Figure 5.7. TCP produced the shortest delay in Scenario 3 and the longest delay in Scenario 1. The delay performance is different from UDP's due to TCP's retransmission mechanism.

Importantly, in CRETP the average delay decreased as the control period shrank. This phenomenon should be understood together with CRETP's conditional retransmission algorithm. CRETP will not disable retransmission until the start of a new

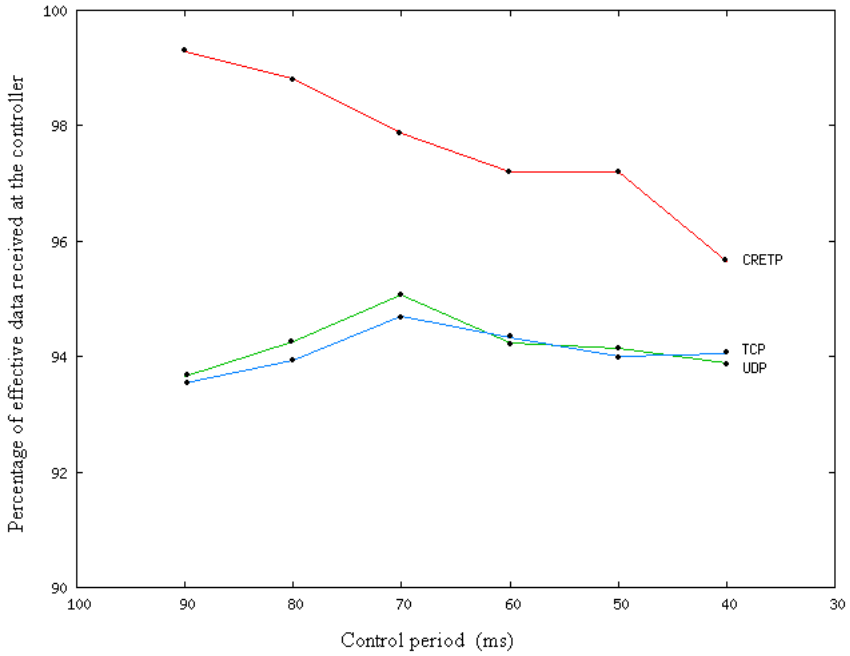

Figure 5.8: Percentage of effective data received in Case Two with different protocols

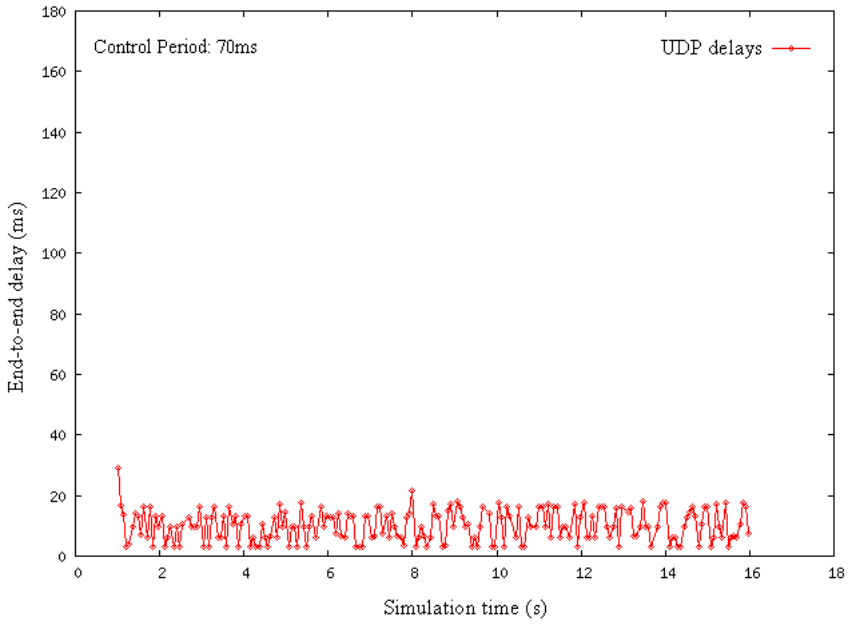

Figure 5.9: End-to-end delays for sensor 1 in Scenario 3 when using UDP

control period with the arrival of new control data. Therefore, retransmission may be conducted more times in the case of a longer control period than in the case of a

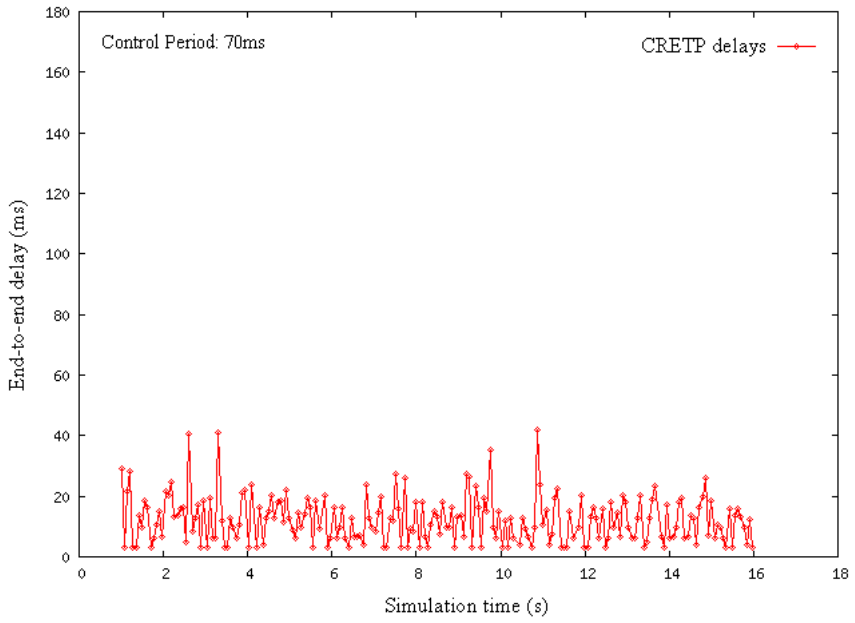

Figure 5.10: End-to-end delays for sensor 1 in Scenario 3 when using CRETP

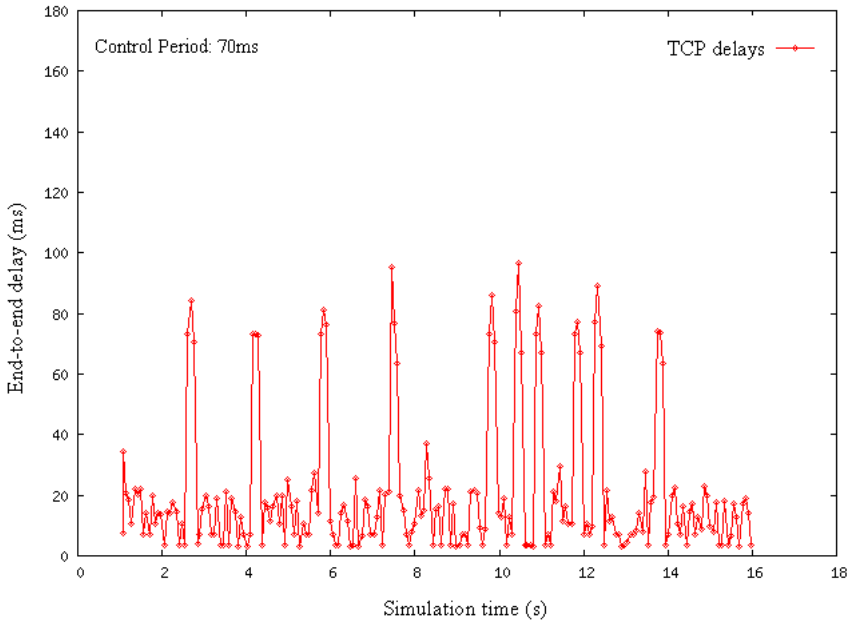

Figure 5.11: End-to-end delays for sensor 1 in Scenario 3 when using TCP

shorter control period. As the opportunities for retransmission grows, CRETP is able to save more data packets from being dropped. Table 5.11 shows the number of dropped data packets in different scenarios. It can be observed that, for CRETP, the longer

the control period is, the fewer are the number of dropped data packets. Only six data packets failed to reach the controller when the control period was 90 milliseconds. The number of dropouts jumps to 82 when the control period is reduced to 40 milliseconds. However, retransmission causes relatively long end-to-end delays. An increase in the average delay is the price paid for a reduction of dropouts. This is the reason why the average delay introduced by CRETP reduces but the number of dropouts increases when the control period shrinks. Fortunately, the average delay increases less than 3 milliseconds when the control period extends from 40 milliseconds to 90 milliseconds. This implies that only 16.7% of a control period is spent on data packet transfer when the control period is 90 milliseconds, while 30.4% of a control period is spent on data packet transfer when the control period is 40 milliseconds. So, the overall performance of CRETP, including delays and reliability, gets better as the control period gets longer.

Delay performances for sensor 1 using different protocols in a randomly chosen scenario, Scenario 3, are graphed in Figures 5.9, 5.10 and 5.11, respectively. These results are similar to the results in Case One. In this scenario, the control period is 70 milliseconds. Many TCP packets have delays even longer than a control period. Both UDP and CRETP provide small and relatively deterministic delays. Only a few delays exceed 40 milliseconds in the case of CRETP.

Table 5.11 lists dropout numbers in the simulations. TCP again distinguishes itself by guaranteeing zero dropouts in all scenarios while UDP still gives the poorest performance with a worst case of 112 dropped packets. The number of dropouts stays stable if the control period is larger than 70 milliseconds, which means that UDP will not perform better in this wireless network model no matter how long the control period is. However, consecutive dropouts happened more frequently with the decrease of the control period. As for CRETP, dropouts shrink dramatically, from 82 to 6, as the control period increases from 40 milliseconds to 90 milliseconds. A larger control period gives CRETP more chances to compensate for data losses. When the control period is larger than 50 milliseconds, there were not any consecutive dropout.

As we know, the ratio of effective data packets received at the controller to the total

number of sent data packets is the proper criterion to define a protocol's transmission reliability. Table 5.13 and Figure 5.8 show the ratios of effective data in the different scenarios. CRETP always provides the most reliable transmission among the three protocols. TCP's advantage in dropouts disappears and it performs no better than UDP.

When examining the occurrence of consecutive losses of effective data at the controller (see Tables 5.14 and 5.15), we found that the occurrence of consecutive dropouts can be avoided if the control period is relatively long. When the control period reduced and reached a certain value, UDP and TCP failed to provide continuous reliable data transmission while CRETP still guaranteed no consecutive data losses at the controller. In terms of maximum number of consecutive data losses, Table 5.15 demonstrates that results provided by CRETP are no worse than those provided by UDP or TCP. Therefore, in our observations, CRETP always performs the best in guaranteeing consecutive successful reliable data transmissions among three protocols when the control period is relatively long, and it does not act worse than UDP and TCP in most cases when the control period gets short.

After analysing the delay and reliability performance of these protocols in Case Two, we can conclude that CRETP has better overall performance than UDP and TCP. With a small increase in the end-to-end delay, but keeping the delay within a certain range, CRETP can significantly improve transmission reliability. Both delay and reliability performances of CRETP become better along with an increase in the control period.

5.5 Simulation Case Three

5.5.1 Simulation Scenario Specifications

In Case Three, we tested protocol performance through adding sensors into the network in the cases of the three control periods. Table 5.16 demonstrates differences between the settings for the nine scenarios in this case study. Scenarios 1 to 3 use the same value of the control period, while the control period in Scenarios 4 to 5 stays fixed.

90 milliseconds is set to be the control period for Scenarios 7 to 9. With the same control period, a larger number of sensors leads to heavier traffic in the network. Based on previous simulation results, we assumed that CRETP may perform slightly worse when more sensors are moved into the network, but can recover if the control period increases.

Table 5.16: Number of sensors and control period in each scenario

Scenario	1	2	3	4	5	6	7	8	9
Number of sensors	5	8	10	5	8	10	5	8	10
Control period (milliseconds)	70	70	70	80	80	80	90	90	90

All scenarios in Case Three were simulated three times. For the first time, the transport layer protocol used was the TCP protocol. Then, UDP was substituted for TCP when all scenarios were tested for the second time. Finally, CRETP acted as the transport protocol in all communications.

5.5.2 Simulation Results

Table 5.17 records the average end-to-end delays for all successfully received data packets in the different scenarios when using different transport layer protocols. The nine scenarios were divided into three groups with scenarios in the same group having the same control period. Figures 5.12, 5.13 and 5.14 are bar graphs that illustrate the average delays in different groups for different protocols. The numbers of dropped data packets in communications are listed in Table 5.18, while Table 5.19 records times of occurrences of consecutive dropouts in each simulation scenarios. Table 5.20 summarises the percentage of effective data packets received at the controller in the different scenarios with different protocols. Numbers of two or more consecutive losses of effective data packets at the controller are recorded in Table 5.21 while Table 5.22 lists maximum number of consecutive data losses in each scenario.

Table 5.17: Average end-to-end delays in Case Three when using different protocols

Scenarios	Control period (milliseconds)	Number of sensors	Average end-to-end delay (milliseconds)		
			UDP	CRETP	TCP
1	70	5	10.055708	12.758372	22.437320
2	70	8	15.297292	19.984371	28.685036
3	70	10	19.504138	26.757464	33.362177
4	80	5	10.131140	13.494351	24.022764
5	80	8	15.405479	20.602708	29.720261
6	80	10	19.550121	27.912518	33.933053
7	90	5	10.037630	15.033824	25.238436
8	90	8	15.366677	21.147166	31.729083
9	90	10	19.520751	26.400619	35.993659

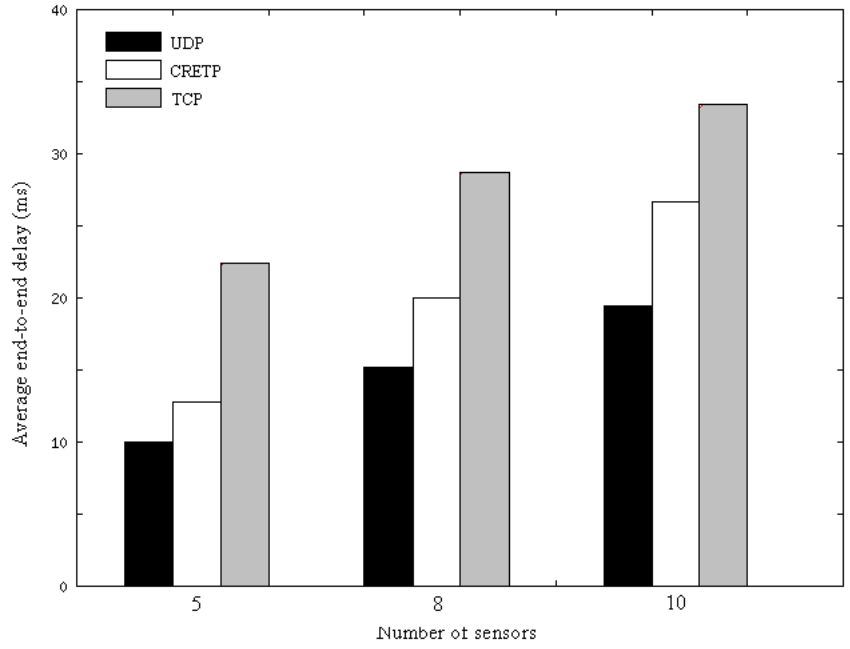

Figure 5.12: Average delays in Case Three with control period of 70 milliseconds

5.5.3 Comparative evaluations

In this case study, we wanted to see how protocol performance would be affected when more sensors are added to the network to communicate with the controller. The number of sensors has three different values in the case study. The minimum value was 5 which is the same as the value in Case Two. The medium value was 8 and the

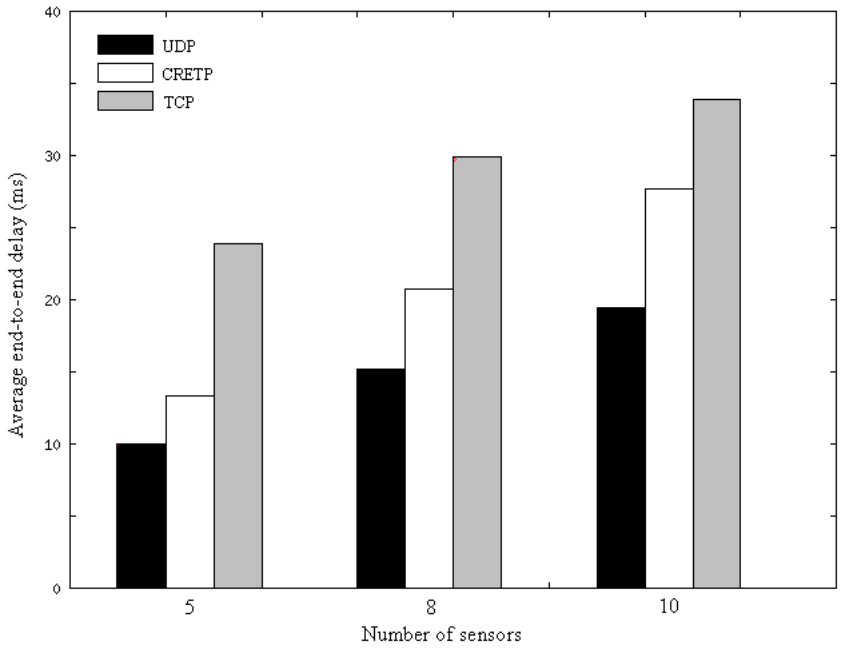

Figure 5.13: Average delays in Case Three with control period of 80 milliseconds

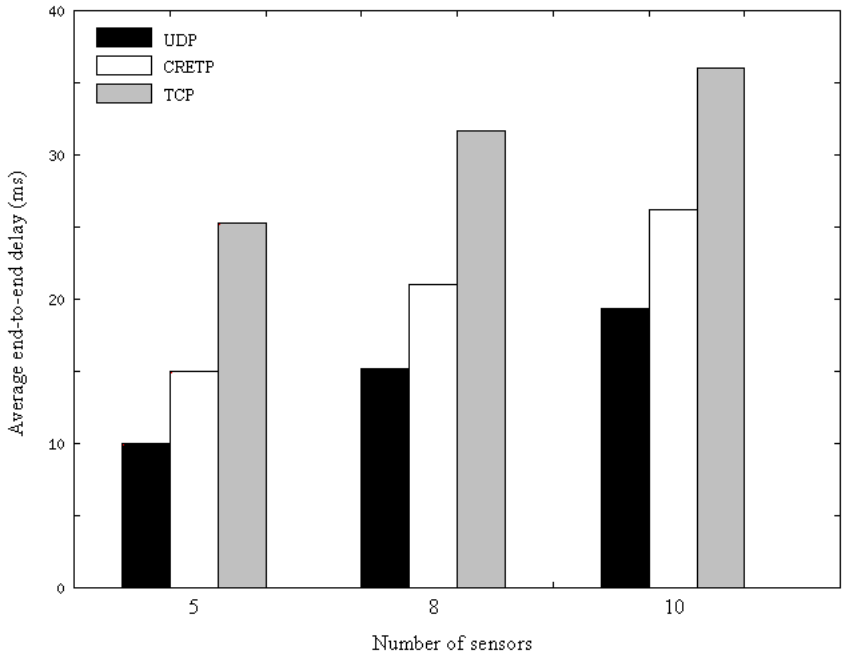

Figure 5.14: Average delays in Case Three with control period of 90 milliseconds

maximum number was 10 which is twice the minimum value. If the control period is kept constant, the number of sending data packets, excluding retransmitted ones,

Table 5.18: Number of dropped data packets in Case Three with different protocols

Scenarios	Control period (milliseconds)	Number of sensors	Number of dropped data packets		
			UDP	CRETP	TCP
1	70	5	53	23	0
2	70	8	99	41	0
3	70	10	123	65	0
4	80	5	54	11	0
5	80	8	86	23	0
6	80	10	108	17	0
7	90	5	53	6	0
8	90	8	82	18	0
9	90	10	104	21	0

Table 5.19: Consecutive dropouts in Case Three when using different protocols

Scenarios	Control period (milliseconds)	Number of sensors	Number of occurrences of consecutive dropouts		
			UDP	CRETP	TCP
1	70	5	2	0	0
2	70	8	7	8	0
3	70	10	12	13	0
4	80	5	2	0	0
5	80	8	1	1	0
6	80	10	2	3	0
7	90	5	0	0	0
8	90	8	0	0	0
9	90	10	3	3	0

doubles when number of sensor changes from 5 to 10. Consequently, the traffic load increases dramatically which may result in network congestion. Unlike the smooth and slow rising of traffic in Case Two, traffic increased sharply and reached a rather high value in Case Three, which makes the delay performance of the three protocols very different. As for UDP and TCP, the average delays no longer fluctuate but keep increasing when the number of sensors rises from 5 to 10. It can be observed from Figures 5.12, 5.13 and 5.14, that UDP still guarantees the smallest average delay while TCP introduces the longest. CRETP performs not as well as UDP with respect to

Table 5.20: Percentage of effective data received in Case Three with different protocols

Scenarios	Control period (milliseconds)	Number of sensors	Percentage of effective data packet		
			UDP	CRETP	TCP
1	70	5	95.070%	97.860%	94.698%
2	70	8	94.244%	97.616%	94.419%
3	70	10	94.279%	96.977%	94.233%
4	80	5	94.255%	98.830%	93.936%
5	80	8	94.282%	98.471%	94.016%
6	80	10	94.256%	99.096%	93.989%
7	90	5	93.653%	99.281%	93.533%
8	90	8	93.862%	98.653%	93.488%
9	90	10	93.772%	98.743%	93.593%

Table 5.21: Consecutive losses of effective data in Case Three with different protocols

Scenarios	Control period (milliseconds)	Number of sensors	Number of occurrences of consecutive losses of effective data		
			UDP	CRETP	TCP
1	70	5	2	0	3
2	70	8	7	8	10
3	70	10	12	13	14
4	80	5	2	0	2
5	80	8	1	1	1
6	80	10	2	3	4
7	90	5	0	0	0
8	90	8	0	0	0
9	90	10	3	3	2

delays; however, the average delays are still small when compared with TCP's. There is no doubt that no matter what protocol is used, delays will continue increasing if there are more sensors entering the network.

When we consider the reliability of the protocols, CRETP shows the best results in all scenarios. Table 5.20 demonstrates that TCP performs even worse than UDP in most scenarios. The smallest rate of effective data packets for CRETP was 96.977% while the best result for UDP was only 95.07%. Table 5.18 lists dropout numbers for the three protocols. Due to CRETP's conditional retransmission scheme, dropped

Table 5.22: Maximum number of consecutive data losses in Case Three with different protocols

Scenarios	Control period (milliseconds)	Number of sensors	Maximum number of of consecutive data losses		
			UDP	CRETP	TCP
1	70	5	0	0	0
2	70	8	3	3	3
3	70	10	3	3	3
4	80	5	0	0	0
5	80	8	3	3	3
6	80	10	3	3	3
7	90	5	0	0	0
8	90	8	0	0	0
9	90	10	3	3	3

packets occur less often when the control period increases. Longer control period also leads to fewer occurrence of consecutive dropout no matter whether UDP or CRETP is used. Generally speaking, CRETP still provides fewer consecutive dropouts than UDP.

In this case study, three protocols provide similar results in terms of consecutive losses of effective data at the controller (see Tables 5.21 and 5.22). They all introduce the same maximum number of consecutive dropouts in simulations. Longer control period leads to fewer occurrences of consecutive dropouts for all three protocols and in that case, CRETP can always guarantee a zero consecutive data loss if there are a small number of sensors in simulations, such as 5 sensors.

Simulation results in this case study show that CRETP performs the best among the three protocols when both the transmission delay and reliability are taken into account. This conclusion is the same as those in Case One and Two. New information found in this case study is that although all protocols perform worse when more working sensors are put into the network, CRETP can recover and even improve its reliability if a longer control period is applied.

5.6 Chapter Summary

Three case studies were conducted to evaluate the performance of CRETP. Comparative studies among CRETP, UDP and TCP were also done. The aim of our simulations was to test if CRETP can provide the best overall performance among the three protocols for real-time applications in an NCS.

Due to UDP's simplicity, it always presents the smallest average end-to-end delay among the three protocols in all simulation scenarios. However, when it comes to reliability, UDP performs much worse than CRETP. There is no doubt that TCP introduces the longest average delay every time. All these large delays are caused by TCP's retransmission algorithm which guarantees zero dropouts in all simulations. Although all data packets can be received by the controller if TCP is used, the received data may be useless and discarded by applications as data timeliness is one of the key issues in a real-time NCS. Therefore, TCP sometimes performs worse than UDP when data effectiveness is considered. There is no doubt that TCP is not suitable for real-time control systems.

As for CRETP, it greatly improves transmission reliability while keeping data timeliness in mind. This implies that CRETP's conditional retransmission algorithm helps guarantee the effectiveness of a successfully retransmitted packet. Since CRETP was designed based on UDP, it maintains UDP's simplicity in some aspects. Therefore, when CRETP is used, the end-to-end delay of a data packet is still a small part of a control period. Although CRETP's delay performance is not as good as UDP's, it still keeps delays at an acceptable level.

After analysing the simulation results, we can positively answer the questions posed at the beginning of this chapter. Our comparative studies demonstrate that only CRETP can guarantee data effectiveness of compensated packets and keep the delay performance at an acceptable level at the same time. These advantages make CRETP a proper transport protocol which can greatly improve the overall performance of real-time applications in an NCS.

Chapter 6

Conclusions and Future Work

Wireless networked control systems (WNCSs) are being increasingly investigated for industrial process control due to their good scalability, fast deployment, and low implementation and maintenance costs. Since a wireless network is not designed with real-time control in mind, it cannot satisfy the communication requirements of NCSs. For instance, a real-time control system requires timely and lossless data transmission. Unfortunately, many distinguishing features of wireless networks, such as channel variation, channel outages and interference, introduce longer transmission delays, bigger jitter and higher rates of packet loss. These network-introduced problems could result in unpredictable system behaviors that lead to system performance degradation or even cause system instability. Therefore, the main objective of our study is to improve network performance in terms of delays, jitters and data loss, so that real-time requirements for WNCSs can be met.

From a networking perspective, modifying existing network protocols, presenting new protocols and introducing appropriate communication schemes, are effective and popular approaches applied in recent research to guarantee real-time performance to a certain extent. One major contribution of our research is the design of a new transport protocol for WNCSs which deals with data dropouts, reducing transmission latency

and jitter, and improving data timeliness. This protocol is named the Conditional Retransmission Enabled Transport Protocol (CRETP), so-called because it includes a conditional retransmission mechanism which significantly improves data transmission reliability. Unacknowledged data packets will be re-transferred by CRETP for a certain amount of times to compensate for data losses. As every data packet in real-time control systems is useful only within a certain deadline, CRETP has the ability to check data effectiveness and guarantee that every data packet delivered to the application layer is valid.

The performance of CRETP was evaluated through extensive simulations, and comparative studies between CRETP, UDP and TCP were also conducted. The results demonstrate that CRETP had the best overall performance for data communications among the three protocols in WNCSs. It provides the most reliable data transmission and maintains relatively small and deterministic delays, which achieves the requirements of real-time control systems in terms of transmission reliability and delays.

From the aspect of network deployment, some strategies about how network layout and the data rate of sensors affect network-introduced latency and data losses were also presented in the thesis. We have evaluated the time delay, jitter and dropout rate for wirelessly connected sensors and controllers for IEEE 802.11b-based UDP/IP NCSs. Solutions for reducing time delays, jitter, and data loss rates were identified through analysis of the simulation results. In particular, when all sensors have the same data rate, the data rate should be chosen with the maximum number of competing sensors in mind; and sensors should not all be placed at the same distance to the controller. However, when sensors have different data rates, slow sensors should be deployed farthest from the controller if there are a large number of sensors communicating in the WNCS.

6.1 Limitations and Future Work

This research work has demonstrated the effectiveness of the proposed communication protocol and network layout strategies in wireless real-time control systems. However, it was found that there were some limitations in the current work that need to be improved and further developments are expected in future studies.

- The proposed protocol was implemented in a network simulator (NS-2), and protocol performance evaluations were done through extensive simulations. However, this is a major limitation of this research project because simulations imitate certain key characteristics and behaviours of a protocol but do not include the full complexity of real systems. In order to see the real-world performance of the protocol, CRETP needs to be implemented into the system kernels of actual devices in WNCSs so that in site measurements of its performance can be conducted.

- For the protocol's performance evaluation, the number of communicating sensors in every simulation scenario was no more than 10. Although it is a distinguishing feature of a control system that the number of interconnecting sensors, controllers, actuators, and other devices is typically low, e.g., a few tens or less, the number of sensors in simulations should be extended to a larger value such as 30. This is because through the simulations conducted so far we found that an increase in the number of communicating devices in a network could have negative effects on both network performance and protocol performance.

  However, we have been unable to get any results when simulating a WNCS with more than 10 sensors each with a practical data rate because the NS-2 simulator's core has always been dumped in our attempts. This is not a problem of the CRETP protocol because simulation of TCP has shown the same problem. It would be worthy to find a way to use NS-2 to simulate WNCSs with a large number of sensors and actuators.

- In this project, the network layout strategies and the new protocol were presented

respectively. Since they were developed from different aspects but are for the same purpose, they can be applied together in a WNCS for system performance improvement.

Further research on a combination of CRETP and proposed network layout strategies should be considered in a future study.

Appendix A

Integrating CRETP inside the NS-2 Simulator

After implementing the CRETP source and destination agents inside NS-2, we need to perform the following steps in order to integrate our code inside the simulator.

A.1 Step one: Add C++ source code into NS-2

To allocate CRETP code we firstly need to create a new directory called cretp inside the NS-2 base directory. All CRETP source code files should be put into this new directory. Four files for CRETP implementation are:

- *cretp-snd.h* This is the header file which defines all necessary methods, timers and source CRETP agents which perform the protocol's functionality.
- *cretp-ack.h* This is the header file which defines all necessary methods, and destination CRETP agents which perform the protocol's functionality.
- *cretp-snd.cc* In this file we implement the timers, source CRETP agents and Tcl hooks.

- *cretp-ack.cc* In this file we implement the destination CRETP agents and Tcl hooks.

A.2 Step two: Tcl library

Some changes have to be done in Tcl files in order to specify the default sizes for our packets. Default values for binded attributes have to be given inside *tcl/lib/ns-default.tcl*. We must go to the end of the file and add something like the next code into the *ns-default.tcl* file:

```
1 Agent/sndCRETP set packetSize_ 512
2 Agent/ackCRETP set packetSize_ 64
```

A.3 Step three: Makefile

To complie the result we must edit the *Makefile* file by adding our object files inside *OBJ_CC* variable as in the following code.

```
1 OBJ_CC = \
2         tools/random.o tools/rng.o tools/ranvar.o common/misc.o common/timer-handler.o \
3         #...
4         cretp/cretp-snd.o cretp/cretp-ack.o \
5         #...
6         $(OBJ_STL)
```

After this, we can execute *make* or run *makeclean* before *make*

Appendix B

C++ source code for CRETP in NS-2

B.1 The header file for the source CRETP

```
1  cretp−snd.h
2
3  #ifndef ns_sndcretp_h
4  #define ns_sndcretp_h
5
6  //include header files required by source CRETP agent
7  #include ”agent.h”
8  #include ”tclcl.h”
9  #include ”packet.h”
10 #include ”address.h”
11
12 #define SAMPLERATE 10000
13
14 class sndCRETPAgent; //forward declaration
15
16 //declare the retransmission timer
17 //this timer class inherits from TimerHandler class
18 //and it has a reference to the source CRETP agent
```

```
19 class OutTimer : public TimerHandler {
20 public:
21         OutTimer(sndCRETPAgent *a) : TimerHandler() { a_ = a; }
22 protected:
23         virtual void expire(Event *e);
24         sndCRETPAgent *a_;
25 };
26
27 //define the source CRETP agent which inherits from Agent class
28 class sndCRETPAgent : public Agent {
29 public:
30         sndCRETPAgent(); //constructor
31         int seq; //sequence number of a CRETP data packet
32         char recordfile[256],agentid[256]; //arrays for results recording
33         int displaystyle,recordstyle; //style of TCL's display
34         double rtt,sendovertime,resendovertime;
35       //rtt is the round-trip-time (ms)
36       //sendovertime represents the sending time of a packet (s)
37       //resendovertime represents the retransmission waiting time (s)
38         int resendnumber,lastpacketsize;
39       //resendnumber recordes how many times a packet has been retransmitted
40         OutTimer out_timer_; // retransmission timer
41         void resendcretpdata(); //function for packet retransmission
42         void sendcretpdata(int newdata, int nbytes, AppData* data);
43       //source CRETP's sending function
44       //when newdata equals to 0, it is a retransmission
45       //when newdata equals to 1, it is a transfer of new data
46         virtual int command(int argc, const char*const* argv);
47       //command() methods that inherits from the Agent class
48         virtual void recv(Packet*, Handler*); //source CRETP's receiving function
49         virtual void sendmsg(int nbytes, AppData* data, const char *flags =0) {
50     sendmsg(nbytes, NULL, flags);
51   } //interface for the application layer to send data to CRETP
52 };
53
```

```
54 #endif
```

B.2 The header file for the destination CRETP

```
1  cretp−ack.h
2
3  #ifndef ns_ackcretp_h
4  #define ns_ackcretp_h
5
6  //include header files required by destination CRETP agent
7  #include ”agent.h”
8  #include ”tclcl.h”
9  #include ”packet.h”
10 #include ”address.h”
11
12 //define the destination CRETP agent which inherits from Agent class
13 class ackCRETPAgent : public Agent {
14 public:
15          ackCRETPAgent(); //constructor
16          int seq; //sequence number of a CRETP data packet
17          virtual int command(int argc, const char∗const∗ argv);
18      //command() methods that inherits from the Agent class
19          virtual void recv(Packet∗, Handler∗);
20      //destination CRETP's receiving function
21 };
22
23 #endif
```

B.3 The C++ file for the source CRETP

```
1  cretp−snd.cc
2
3  #include ”rtp.h”
4  #include ”fstream.h”
5  #include ”CRETP−snd.h”
```

```
6
7  //Tcl hooks: bind the source CRETP agent class to Tcl interface
8  static class sndCRETPClass : public TclClass {
9  public:
10             sndCRETPClass() : TclClass("Agent/sndCRETP") {}//class constructor
11             TclObject* create(int, const char*const*) {
12               return (new sndCRETPAgent());
13    //implement a function called create() which returns a new sndCRETPAgent instance as a TclObject
14             }
15 } class_sndCRETP;
16
17 //constructor implementation
18 sndCRETPAgent::sndCRETPAgent() : Agent(PT_UDP), seq(-1), rtt(10), displaystyle(0) , recordstyle
       (0) , out_timer_(this)
19  //PT_UDP is the packet type
20  //rtt is the round-trip-time (ms)
21  //seq represents the sequence number of a CRETP data packet
22  //displaystyle indicates how TCL displays simulation results
23  //recordstyle indicates how TCL records simulation results
24  //create the retransmission timer: out_timer_
25 {
26    bind("packetSize_", &size_);
27    //bind packetSize_ as an integer which now may be read and written from Tcl
28 }
29
30 //retransmission function implementation
31 void sndCRETPAgent::resendcretpdata()
32 {
33    sendcretpdata(0,lastpacketsize,lastpacketdata);
34    //call the packet sending function and specify it is the transmission of an old data packet as newdata
35    //equals to 0
36 }
37
38 //packet sending function implementation
39 void sndCRETPAgent::sendcretpdata(int newdata, int nbytes, AppData* data)
```

```
40 {
41   if (newdata) { //do transmission of new data when newdata equals to 1
42     seq++;
43     resendnumber=0;
44     lastpacketsize=nbytes; //packet size
45     lastpacketdata=data; //application layer data
46   }
47   else //do retransmission of an old data packet
48   {
49     resendnumber++;
50     if (resendnumber<=3) rtt*=2;
51     //retransmission time doubles if a packet has not been retransmitted for more than three times
52     }
53
54 Packet* pkt = allocpkt(); //create a CRETP packet
55 if (nbytes<=0) nbytes=size_;
56 hdr_cmn::access(pkt)->size() = nbytes; //record packet size
57 hdr_rtp* hdr = hdr_rtp::access(pkt); //access packet header
58 hdr->flags() = 0; //fill out the flag field
59 hdr->seqno() = seq; //fill out the sequence number field
60 hdr_cmn::access(pkt)->timestamp()=(u_int32_t)(SAMPLERATE*Scheduler::instance().clock());
61 //fill out the timestamp field with current time
62 pkt->setdata(data); //encapsulate data from application layer
63 send(pkt, 0); //send packet
64 resendovertime=(rtt/1000.0)*2; //set retransmission time
65 out_timer_.resched(resendovertime); //start the retransmission timer
66
67   if (displaystyle==1){
68     //Tcl display the sequence number and the sending time of a data packet
69     //and the current round-trip-time if displaystyle equals to 1
70         char out[100];
71   if (newdata)
72     sprintf(out, "puts \"sendnew seq=%d rtt=%.1fms time=%3.1fms\"",seq,rtt,Scheduler::instance().
          clock()*1000);
73   else
```

```
74        sprintf(out, ”puts \”sendold seq=%d rtt=%.1fms time=%3.1fms\””,seq,rtt,Scheduler::instance().
              clock()∗1000);
75     Tcl& tcl = Tcl::instance();//obtain a reference to the class Tcl instance
76     tcl.eval(out); //invoke Tcl_GlobalEval() to execute out
77     }
78
79   if ( (recordstyle==1 && !newdata) || (recordstyle==2 && newdata) || (recordstyle==3) ) {
80    //Tcl records information for retransmitted packets
81    char out[100];
82    if (newdata)
83    sprintf(out, ”%s sendnew seq=%d rtt=%.1fms time=%3.1fms kind=cbr”,agentid,seq,rtt,Scheduler::
          instance().clock()∗1000);
84    else
85    sprintf(out, ”%s sendold seq=%d rtt=%.1fms time=%3.1fms kind=cbr”,agentid,seq,rtt,Scheduler::
          instance().clock()∗1000);
86    ofstream filerecord;
87    filerecord.open(recordfile,ios::app);
88    filerecord<<out<<endl;
89    filerecord.close();
90    }
91   }
92
93   //implementation of command() methods that inherits from the Agent class
94   int sndCRETPAgent::command(int argc, const char∗const∗ argv){
95     //argv[0] contains the name of the method being invoked
96     //argv[1] is the requested operation
97     //argv[2..argc−1] are the rest of the arguments which were passed
98    if (argc == 3) //the case where there are three arguments{
99     if (strcmp(argv[1], ”display”) == 0) {
100      if (strcmp(argv[2], ”screen”) == 0) displaystyle=1;
101        //Tcl displays according to the first sytle
102      return (TCL_OK); //return with success
103      }
104      if (strcmp(argv[1], ”clear”) == 0) {
105      sprintf(recordfile,”%s”,argv[2]);//erase records in the specified file
```

```
106       ofstream fileclear(recordfile);
107       fileclear.close();
108       return (TCL_OK); //return with success
109       }
110     }
111    if (argc == 5) //the case where there are five arguments{
112     if (strcmp(argv[1], "record") == 0) {
113       sprintf(recordfile,"%s",argv[2]);
114       sprintf(agentid,"%s",argv[3]); //agented records the id of sending node
115       if (strcmp(argv[4], "resend") == 0) recordstyle=1;
116         //Tcl records information of retransmitted packets into a file
117       if (strcmp(argv[4], "send") == 0) recordstyle=2;
118       //Tcl records information of packets sent for the first time into a file
119       if (strcmp(argv[4], "all") == 0) recordstyle=3;
120         //Tcl records information of packets sent every time into a file
121       return (TCL_OK); //return with success
122       }
123     }
124    return (Agent::command(argc, argv));
125       //delegate the responsibility to base class if the requested command
126       //is not processed by sndCRETPAgent()::command
127  }
128
129  //function for receiving data from the application layer
130  //it is invoked whenever the source CRETP agent receives a packet from the application layer
131  void sndCRETPAgent::sendmsg(int nbytes, AppData* data, const char* flags){
132    int n;
133    if (size_) //divide application data if it is too big for one CRETP packet
134      n = nbytes / size_;
135    else
136      printf("Error: sndCRETP size = 0\n");
137    if (nbytes == -1) {
138      printf("Error: sendmsg() for sndCRETP should not be -1\n");
139      return;
140      }
```

```
141   if (data && nbytes > size_) {
142     printf("Error: data greater than maximum sndCRETP packet size\n");
143     return;
144     }
145   while (n-- > 0) sendcretpdata(1,size_,data); //call sendcretpdata() to send data
146   n = nbytes % size_;
147   if (n > 0) sendcretpdata(1,n,data);
148   idle();
149 }
150
151 //source CRETP's receiving function for ACKs
152 //it is invoked whenever the source CRETP agent receives a packet
153 void sndCRETPAgent::recv(Packet* pkt, Handler*)
154 {
155     hdr_rtp* hdr = hdr_rtp::access(pkt); //access packet header
156     if ((hdr->flags()!= 1) || (hdr->seqno() != seq)) return; //check if it is the wanted ACK
157     rtt=rtt*7/8.0 + (Scheduler::instance().clock()-hdr_cmn::access(pkt)->timestamp()*1.0/
            SAMPLERATE)*1000/8.0
158     //updata RTT: rtt=rttold*7/8+rttnew*1/8
159     out_timer_.force_cancel(); //cancel the retransmission timer
160       if (displaystyle==1) //TCL displays according to display style 1
161       {
162         char out[100];
163         sprintf(out, "puts \"receive seq=%d rtt=%.1fms time=%3.1fms\"",hdr->seqno(),rtt,Scheduler::
                instance().clock()*1000);
164         Tcl& tcl = Tcl::instance();
165           //obtain a reference to the class Tcl instance
166         tcl.eval(out); //invoke Tcl_GlobalEval() to execute out
167       }
168       Packet::free(pkt); //free ACK
169 }
170
171 //expire() method for the retransmission timer
172 void OutTimer::expire(Event*) {
173     a_->resendcretpdata(); //enable retransmission when a time_out happens
```

```
174 }
```

B.4 The C++ file for the destination CRETP

```
1  cretp−ack.cc
2
3  #include ”rtp.h”
4  #include ”cretp−ack.h”
5
6  //Tcl hooks: bind the destination CRETP agent class to Tcl interface
7  static class ackCRETPClass : public TclClass {
8  public:
9     ackCRETPClass() : TclClass(”Agent/ackCRETP”) {} //class constructor
10    TclObject∗ create(int, const char∗const∗) {
11     return (new ackCRETPAgent());
12 //implement a function called create() which returns a new ackCRETPAgent instance as a TclObject
13    }
14 } class_ackcretp;
15
16 //constructor implementation
17 ackCRETPAgent::ackCRETPAgent() : Agent(PT_ACK), seq(0)
18  //packet type is PT_ACK
19  //seq represents the sequence number of a CRETP data packet
20 {
21     bind(”packetSize_”, &size_);
22     //bind packetSize_ as an integer which now may be read and written from Tcl
23 }
24
25 //implementation of command() methods that inherits from the Agent class
26 int ackCRETPAgent::command(int argc, const char∗const∗ argv){
27 return (Agent::command(argc, argv)); //delegate the responsibility to base class
28 }
29
30 //destiantion CRETP's receiving function for data packets
31 //it is invoked whenever the destination CRETP agent receives a packet
```

```
32 void ackCRETPAgent::recv(Packet* pkt, Handler*) {
33   hdr_rtp* hdr = hdr_rtp::access(pkt); //access packet header
34   if (hdr->flags()!= 0) return; //check if it is a data packet
35   if (hdr->seqno() < seq) return; //check data effectiveness
36   u_int32_t sendtime = hdr_cmn::access(pkt)->timestamp();
37   //acquire the sending time of this data packet
38   int rcv_seq = hdr->seqno(); //acquire the sequence number of this data packet
39   seq = rcv_seq + 1; //set the expected sequence number for the next time
40   if (app_) { //process user data
41     hdr_cmn* hdrcmn = hdr_cmn::access(pkt);
42     app_->process_data(hdrcmn->size(), pkt->userdata());
43   }
44   Packet::free(pkt); //free this packet
45
46   Packet* pktret = allocpkt();//create corresponding ACK
47   hdr_rtp* hdrret = hdr_rtp::access(pktret);//access ACK header
48   hdrret->flags() = 1; //ACK's flag field should be 1
49   hdrret->seqno() = rcv_seq; //set ACK's sequence number
50   hdr_cmn::access(pktret)->timestamp() = sendtime;
51   //save the sending time of data packet for RTT calculation
52   send(pktret, 0); //send ACK
53 }
```

Appendix C

Example Tcl script for simulation in NS-2

Here is one of the Tcl scripts for the simulations in Chapter 5. It gives an example of how CRETP is used in NS-2 simulations. This script corresponds to the scenario of 10 sensors with the same control period of 80 milliseconds.

```
1  10s80msCretp.tcl
2
3  # simulation of ten sensors and a controller
4  # define options
5  set val(chan) Channel/WirelessChannel ;# channel type
6  set val(prop) Propagation/TwoRayGround ;# radio−propagation model
7  set val(netif) Phy/WirelessPhy ;# network interface type
8  set val(mac) Mac/802_11 ;# MAC layer type
9  set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
10 set val(ll) LL ;# link layer type
11 set val(ant) Antenna/OmniAntenna ;# antenna model
12 set val(ifqlen) 50 ;# max packet in ifq
13 set val(bandwidth) 1.0e6 ;# bandwidth of network
14 set val(nn) 11 ;# number of mobilenodes
```

```
15 set val(rp) DSR ;# routing protocol
16 set val(x)  300 ;# x dimension
17 set val(y)  300 ;# y dimension
18 set val(r) 50 ;# radius of the circle
19 set val(tr) w.tr ;# trace file
20 set val(nam) w.nam ;# nam trace file
21 set val(procstart) 1.99 ;# first call of detach proc
22 set val(initialdetach) 2.00 ;# first time of breakdown
23 set val(procendafter) 15.00 ;# end time of unstable connection
24 set val(basic) 0.45 ;# basic time period between breakdowns
25 set val(stop) 18 ;# simulation time
26
27 # other default settings (optional)
28 LL set bandwidth_ 0
29 Agent/Null set sport_ 0
30 Agent/Null set dport_ 0
31 Agent/CBR set sport_ 0
32 Agent/CBR set dport_ 0
33 Agent/TCPSink set sport_ 0
34 Agent/TCPSink set dport_ 0
35 Agent/TCP set sport_ 0
36 Agent/TCP set dport_ 0
37 Queue/DropTail/PriQueue set Prefer_Routing_Protocols 1
38
39 # unity gain, omni−directional antennas
40 # set up the antennas to be centered in the node and 1.5 meters above it
41 Antenna/OmniAntenna set X_ 0
42 Antenna/OmniAntenna set Y_ 0
43 Antenna/OmniAntenna set Z_ 1.5
44 Antenna/OmniAntenna set Gt_ 1.0
45 Antenna/OmniAntenna set Gr_ 1.0
46
47 # initialize the SharedMedia interface with parameters to make it work
48 # like the 914MHz Lucent WaveLAN DSSS radio interface
49 Phy/WirelessPhy set CPThresh_ 10.0
```

```
50  Phy/WirelessPhy set CSThresh_ 1.559e−11
51  Phy/WirelessPhy set RXThresh_ 3.652e−10
52  Phy/WirelessPhy set Rb_ 2∗1e6
53  Phy/WirelessPhy set Pt_ 0.2818
54  Phy/WirelessPhy set freq_ 914e+6
55  Phy/WirelessPhy set L_ 1.0
56  Phy/WirelessPhy set bandwidth_ 1.0e6
57  Mac/802_11 set dataRate_ 1.0e6
58  Mac/802_11 set PLCPDataRate_ 1.0e6
59  Mac/802_11 set basicRate_ 1.0e6
60
61  # Get random numbers from files
62  # rdm1 is for start time; rdm2 is for down period
63  set infile1 [open ”rdarray1.dat” r]
64  set infile2 [open ”rdarray2.dat” r]
65  gets $infile1 rdmnum1
66  set number_1 [llength $rdmnum1]
67  gets $infile2 rdmnum2
68  set number_2 [llength $rdmnum2]
69  for {set i 0} {$i < $number_1} {incr i} {
70      set rdm1_($i) [lindex $rdmnum1 $i]
71  }
72  for {set i 0} {$i < $number_2} {incr i} {
73      set rdm2_($i) [lindex $rdmnum2 $i]
74  }
75
76  # main program
77  # initialize global variables and create simulator instance
78  set ns_  [new Simulator]
79
80  set topo [new Topography] # setup topography object
81  $ns_ use−newtrace # create trace object for ns
82  set tracefd [open $val(tr) w]
83  $ns_ trace−all $tracefd
84
```

```
85  $topo load_flatgrid $val(x) $val(y) # define topology
86  create-god $val(nn) # Create God
87
88  # global node setting, configure node
89  $ns_ node-config -adhocRouting $val(rp) \
90                   -llType $val(ll) \
91                   -macType $val(mac) \
92                   -ifqType $val(ifq) \
93                   -ifqLen $val(ifqlen) \
94                   -antType $val(ant) \
95                   -propType $val(prop) \
96                   -phyType $val(netif) \
97                   -topoInstance $topo \
98                   -agentTrace ON \
99                   -routerTrace ON \
100                  -macTrace ON \
101                  -movementTrace OFF \
102                  -channel $chan_1_
103
104 for {set i 0} {$i < $val(nn) } {incr i} {
105         set node_($i) [$ns_ node]
106         $node_($i) random-motion 0  ;# disable random motion
107 }
108
109 # provide initial (X,Y, for now Z=0) co-ordinates for wireless nodes
110 # controller
111 $node_(0) set X_ 100.0
112 $node_(0) set Y_ 100.0
113 $node_(0) set Z_ 0.0
114
115 # sensors
116 $node_(1) set X_ 150.0
117 $node_(1) set Y_ 100.0
118 $node_(1) set Z_ 0.0
119
```

```
120 $node_(2) set X_ 100.0
121 $node_(2) set Y_ 150.0
122 $node_(2) set Z_ 0.0
123
124 $node_(3) set X_ 50.0
125 $node_(3) set Y_ 100.0
126 $node_(3) set Z_ 0.0
127
128 $node_(4) set X_ 100.0
129 $node_(4) set Y_ 50.0
130 $node_(4) set Z_ 0.0
131
132 $node_(5) set X_ 60.0
133 $node_(5) set Y_ 70.0
134 $node_(5) set Z_ 0.0
135
136 $node_(6) set X_ 60.0
137 $node_(6) set Y_ 130.0
138 $node_(6) set Z_ 0.0
139
140 $node_(7) set X_ 140.0
141 $node_(7) set Y_ 70.0
142 $node_(7) set Z_ 0.0
143
144 $node_(8) set X_ 140.0
145 $node_(8) set Y_ 130.0
146 $node_(8) set Z_ 0.0
147
148 $node_(9) set X_ 130.0
149 $node_(9) set Y_ 60.0
150 $node_(9) set Z_ 0.0
151
152 $node_(10) set X_ 130.0
153 $node_(10) set Y_ 140.0
154 $node_(10) set Z_ 0.0
```

```
155
156 # Define traffic model
157 puts ”Setting traffic connection...”
158
159 # CRETP and CBR connections between nodes
160 for {set i 1} {$i < $val(nn) } {incr i}
161 {
162   set sndcretp_($i) [new Agent/sndCRETP]
163   $ns_ attach−agent $node_($i) $sndcretp_($i)
164
165   set ackcretp_($i) [new Agent/ackCRETP]
166   $ns_ attach−agent $node_(0) $ackcretp_($i)
167
168   $ns_ connect $sndcretp_($i) $ackcretp_($i)
169   $ns_ at 1.0 ”$sndcretp_($i) record rtt_($i).tr sndcretp_($i) all”
170
171   set cbr_($i) [new Application/Traffic/CBR]
172   $cbr_($i) set packetSize_ 200
173   $cbr_($i) set interval_ 0.08s
174   $cbr_($i) set random_ 0
175   $cbr_($i) attach−agent $sndcretp_($i)
176   $ns_ at 1.0 ”$cbr_($i) start”
177   $ns_ at 16.0 ”$cbr_($i) stop”
178 }
179
180 set timesofdetach 0
181 set n 1
182 set m 30
183 set detach_time $val(initialdetach)
184 $ns_ at $val(procstart) ”de_and_reattach”
185 proc de_and_reattach {} {
186     global ns_ detach_time node_ n m val timesofdetach ackcretp_ rdm1_ rdm2_
187     for {set i 1} {$i<$val(nn)} {incr i} {
188         $ns_ at $detach_time ”$ns_ detach−agent $node_(0) $ackcretp_($i)”
189     }
```

```
190     set current_time [$ns_ now]
191     set retach_time [expr $detach_time+$rdm2_($n)]
192     for {set i 1} {$i<$val(nn)} {incr i} {
193         $ns_ at $retach_time "$ns_ attach-agent $node_(0) $ackcretp_($i)"
194     }
195     set next_time [expr $val(basic)* [expr 1.0 + $rdm1_($m)]]
196     set detach_time [expr $next_time+$current_time]
197     $ns_ at $detach_time "if {$detach_time < $val(procendafter)} {de_and_reattach}"
198     set timesofdetach [expr 1+$timesofdetach]
199     set n [expr 1+$n]
200     set m [expr 1+$m]
201 }
202
203 # define node initial position in nam
204 for {set i 0} {$i < $val(nn)} {incr i} {
205  # 20 defines the node size in nam, must adjust it according to your scenario
206  # the function must be called after mobility model is defined
207     $ns_ initial_node_pos $node_($i) 20
208 }
209
210 # tell nodes when the simulation ends
211 for {set i 0} {$i < $val(nn) } {incr i} {
212     $ns_ at $val(stop).0001 "$node_($i) reset";
213 }
214
215 $ns_ at $val(stop).0000 "finish"
216 proc finish {} {
217     global ns_ tracefd
218     $ns_ flush-trace
219     close $tracefd
220     exit 0
221 }
222
223 $ns_ at $val(stop).0001 "puts \"NS EXITING...\" ; $ns_ halt"
224 puts "Starting Simulation..."
```

```
225 $ns_ run
226 }
```

Bibliography

Afonso, J. A. and Neves, J. E. (2004). Scheduling of real-time traffic in IEEE 802.11 networks. In *European conference on the use of modern information and communication technologies*, pages 113–120, Ghent, Belgium.

Afonso, J. A. and Neves, J. E. (2005). Fast retransmission of real-time traffic in HIPERLAN/2 system. In *Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop*, pages 34–38, Lisbon, Portugal.

Altman, E. and Jimenez, T. (2003). NS simulator for beginners.

Baillieul, J. and Antsaklis, P. J. (2007). Control and communication challenges in networked real-time systems. *Proceedings of the IEEE*, 95(1):9–28.

Baldwin, R. O., IV, N. J. D., and Midkiff, S. F. (1999). A real-time medium access control protocol for ad hoc wireless local area networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(2):20–27.

Bianchi, G., Fratta, L., and Oliveri, M. (1996). Performance evaluation and enhancement of the CSMA/CA MAC protocol for 802.11 wireless LANs. In *Seventh IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'96)*, volume 2, pages 392–396, Taipei, Taiwan.

Bova, T. and Krivoruchk, T. (1999). RELIABLE UDP PROTOCOL.

Cena, G., Bertolotti, I. C., Valenzano, A., and Zunino, C. (2007). Evaluation of

response times in industrial WLANs. *IEEE Transactions on Industrial Informatics*, 3(3):191–201.

Cena, G., Valenzano, I. C. B. A., and Zunino, C. (2008). Industrial applications of IEEE 802.11e WLANs. In *IEEE international Workshop on Factory Communication Systems (WFCS08)*, pages 129–138, Dresden, Germany.

Cervin, A., Arzen, K. E., Henriksson, D., Lluesma, M., Balbastre, P., Ripoll, I., and Crespo, A. (2006). Control loop timing analysis using truetime and jitterbug. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 1194–1199, Munich, Germany.

Cervin, A., Henriksson, D., Lincoln, B., Eker, J., and Arzen, K. E. (2003). How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *Control Systems Magazine, IEEE*, 23(3):16–30.

Cheng, C. W., Lai, C. L., Wang, B. C., and Hsu, P. L. (2007). The time-delay effect of multiple-network systems in NCS. In *SICE Annual Conference 2007*, pages 929–934, Takamatsu, Kagawa, Japan.

Cloosterman, M., van de Wouw, N., Heemels, W. P. M. H., and Nijmeijer, H. (2009). Stability of networked control systems with uncertain time-varying delays. *Automatic Control, IEEE Transactions on*, 54(7):1575–1580.

Cloosterman, M. B. G., Wouw, N. V. D., Heemels, W. P. M. H., and Nijmeijer, H. (2008). Stabilization of networked control systems with large delays and packet dropouts. In *2008 American Control Conference*, pages 4991–4996, Seattle, Washington, USA.

Cunningham, R. and Cahill, V. (2002). Time bounded medium access control for ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 1–8, Toulouse, France.

Deng, D. J. and Chang, R. S. (1999). A priority scheme for IEEE802.11 DCF access method. *IEICE Transactions on Communications (Inst Electron Inf Commun Eng)*, E82-B(1):96–102.

Dermanovic, B., N. Peric, N., and Petrovic, I. (2004). Modeling of transport delay on Ethernet communication networks. In *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference (MELECON 2004)*, volume 1, pages 367–370, Dubrovnik, Croatia.

Forouzan, B. A. and Fegan, S. C. (2003). TCP/IP protocol suite. In Lupash, E. J., editor, *McGraw-Hill Forouzan networking series*. E. A. Jones, second edition.

Groups, U. (2007). The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/doc/, Accessed on 28 June, 2007.

Gupta, V., Spanos, D., Hassibi, B., and Murray, R. (2007). Optimal LQG control across packet-dropping link. *Systems & Control Letters*, 56(6):439–446.

He, W. H., Ge, Z. H., and Hu, Y. P. (2007). Optimizing UDP packet sizes in ad hoc networks. In *International Conference on Wireless Communications, Networking and Mobile Computing (WiCom2007)*, pages 1617–1619, Shanghai, P. R. China.

Hespanha, J. P., Naghshtabrizi, P., and Xu, Y. G. (2007). A survey of recent results in networked control systems. *Proceedings of the IEEE*, 95(1):138 – 162.

Hirano, Y. and Murase, T. (2008). Evaluation of packet loss effect on throughput unfairness between TCP upflows over IEEE 802.11 wireless LAN. In *7th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT2008)*, pages 258–262, Bandos Island.

Hristu-Varsakelis, D. and Levine, W. S. (2005). *Handbook of Networked and Embedded Control Systems (Control Engeering)*. Birkhauser, Boston, New York, USA. R. Alur and K. -E Arzen and J. Baillieul and T. A. Henzinger.

Hsu, M. C. and Chen, Y. C. (2006). Enhanced PCF Protocols for Real-time Multimedia Services over 802.11 Wireless Networks. In *Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*, pages 56–56, Lisboa, Portugal.

Itaya, S., Kosuga, M., and Davis, P. (2004). Evaluation of packet latency and fluctuation during UDP packet exchange in ad hoc wireless groups. In *Proceedings of the 24th internation conference on Distributed Computing Systems Workshops (ICDCSW'04)*, pages 684–689, Hachioji, Tokyo, Japan.

Jiang, S. (1998). Wireless communications and a priority access protocol for multiple mobile terminals in factory automation. *IEEE Transactions on Robotics and Automation*, 14(1):137–143.

John, S. N., Ibikunle, F. A., and Adewale, A. A. (2008). Performance improvement of wireless network based on effective data transmission. In *IET International Conference on Wireless, Mobile and Multimedia Networks*, pages 134–137, Mumbai, Inidia.

Jonsson, M. and Kunert, K. (2008). Meeting reliability and real-time demands in wireless industrial communication. In *The 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2008)*, pages 877–884, Hamburg, Germany.

Khayam, S. A., Karande, S., Krappel, M., and Radha, H. (2003a). Cross-layer protocol design for real-time multimedia applications over 802.11 b networks. In *Proceedings of 2003 International Conference on Multimedia and Expo*, volume 2, pages 425–428, Baltimore, Maryland, USA.

Khayam, S. A., Karande, S., Radha, H., and Loguinow, D. (2003b). Performance analysis of errors and losses over 802.11b LANs for High-Bitrate Real-Time Multimedia. *Signal Processing: Image Communication*, 18(7):575–595.

Lam, P. P. K. and Liew, S. C. (2004). UDP-Liter: an improved UDP protocol for real-time multimedia applications over wireless links. In *The 1st International Symposium on Wireless Communication Systems (ISWCS2004)*, pages 314–318, Mauritius.

Larzon, L., Degermark, M., and Pink, S. (1999). UDP-Lite for Real Time Multimedia Applications.

Larzon, L., Degermark, M., and Pink, S. (2004). The Lightweight User Datagram Protocol (UDP-Lite).

Le, T., Kuthethoor, G., Hansupichon, C., Sesha, P., Strohm, J., Hadynski, G., Kiwior, D., and Parker, D. (2009). Reliable User Datagram Protocol for airborne network. In *IEEE Military Communications Conference (MILCOM2009)*, pages 1–6, Boston, Massachusetts, USA.

Lee, J. Y., Kim, G. Y., and Park, S. K. (2002). Optimum UDP packet sizes in ad hoc networks. In *2002 Workshop on High Performance Switching and Routing*, pages 214–218, Kobe, Hyogo, Japan.

Li, Q. and Yao, C. (2003). *Real-Time Concepts for Embedded Systems*. CMP Books. editor: R. Ward and M. Briand.

Liu, P. X., Meng, M., Ye, X. F., and Gu, J. (2002). An UDP-based protocol for internet robots. In *Proceedings of the 4th World Congress on Intelligent Control and Automation*, volume 1, pages 59–65, Shanghai, P.R.China.

Liu, X. and Goldsmith, A. (2004). Wireless medium access control in networked control systems. In *Proceedings of the 2004 American Control Conference*, volume 4, pages 3605–3610, Boston, Massachusetts, USA.

Loeser, J. and Haertig, H. (2004). Low-latency hard real-time communication over switched Ethernet. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 2004)*, pages 13–22, Catania, Sicily, Italy.

Lu, H. W. (2001). How to process the packet lost and out of order based on UDP transport protocol. *Computer Engineering and Applications*, 2:48–50.

Luck, R. and Ray, A. (1990). An observer-based compensator for distributed delays. *Automatica, IFAC*, 26(5):903–908.

Luck, R. and Ray, A. (1994). Experimental verification of a delay compensation algorithm for integrated communication and control systems. *International Journal of Control*, 59(6):1357–1372.

Markowski, M. J. and Sethi, A. S. (1998). Fully distributed wireless MAC transmission of real-time data. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium*, pages 49–57, Denver, Colorado, USA.

Nilsson, J. (1998). *Real-time control systems with delays*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Ouni, S. and Kamoun, F. (2002). Hard and soft real time message scheduling on Ethernet networks. In *2002 IEEE International Conference on Systems, Man and Cybernetics*, volume 6, pages 378–382, Hammamet, Tunisia.

Pal, A., Dogan, A., and Ozguner, F. (2002a). MAC layer protocols for real-time traffic in ad-hoc wireless networks. In *Proceedings of the 2002 International Conference on Parallel Processing (ICPP2002)*, pages 539 – 546, Vancouver, British Columbia, Canada.

Pal, A., Dogan, A., Ozguner, F., and Ozguner, U. (2002b). A MAC layer protocol for real-time inter-vehicle communication. In *Proceedings of the IEEE 5th International Conference on Intelligent Transportation Systems*, pages 353–358.

Salyers, D. C., Striegel, A. D., and Poellabauer, C. (2008). Wireless reliability: Rethinking 802.11 packet loss. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM2008)*, pages 1–4, Newport Beach, CA, U.S.A.

Shakkottai, S., Rappaport, T., and Karlsson, P. (2003). Cross-layer design for wireless networks. *IEEE Communications Magazine*, 41(10):74–80.

Singh, A., Konrad, A., and Joseph, A. D. (2001). Performance evaluation of UDP Lite for cellular video. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 117–124, Port Jefferson, New York, United States.

Stoina, P. (2008). Transport protocol for a real-time communication in wireless sensor actor networks. Master's thesis, Genie Informatique, Networking,Universite Montreal.

Tian, Y. C., Levy, D., Tade, M. O., Gu, T. L., and Fidge, C. (2006). Queuing packets in communication networks for networked control systems. In *The 6th World Congress on Intelligent Control and Automation (WCICA'06)*, volume 1, pages 205–209, Dalian, P. R. China.

Tipsuwan, Y. and Chow, M. Y. (2003). Control methodologies in networked control systems. *Control Engineering Practice*, 11(10):1099–1111.

Tsigkas, O. and Pavudou, F. N. (2008). Providing QOS support at the distributed wireless MAC layer: a comprehensive study. *Wireless Communications, IEEE*, 15(1):22–31.

Wang, L. and Zhen, K. (2010). Performance analysis of reliable dynamic buffer UDP over wireless networks. In *Second International Conference on Computer Modeling and Simulation (ICCMS '10)*, volume 1, pages 114–117, Sanya, Hainan, P. R. China.

Wang, X., Wang, S., and Jiang, A. G. (2006). Optimized deployment strategy of mobile agents in wireless sensor networks. In *Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)*, volume 2, pages 893–898, Jinan, Shandong, P.R.China.

Watteyne, T. and Auge-Blum, I. (2005). Proposition of a hard real-time MAC protocol for wireless sensor networks. In *13th IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, page 533 536, Atlanta, Georgia, USA.

Wen, P., Cao, J., and Li, Y. (2007). Design of high-performance networked real-time control systems. *Control Theory and Applications, IET*, 1(5):1329–1335.

Wittenmark, B., Nilsson, J., and Torngren, M. (1995). Timing problems in real-time control systems. In *Proceedings of the American Control Conference*, volume 3, pages 2000–2004, Seattle, Washington, USA.

Wu, C. X. (2006). Practical models and control methods with data packets loss on NCS. In *2006 IET International Conference on Wireless, Mobile and Multimedia Networks*, pages 1–4, Hangzhou, P. R. China.

Wu, J. C. S. (2001). A real time transport scheme for wireless multimedia communications. *Mobile Networks and Applications*, 6(6):535–546.

Yang, Y. and Wang, Y. J. (2005). Modeling and control for NCS with time-varying long delays. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1407–1411, Guangzhou, P. R. China.

Ye, H. (2000). *Research on networked control systems*. PhD thesis, Department of Mechanical Engineering, University of Maryland, USA.

Ye, H., Walsh, G., and Bushnell, L. G. (2001). Real-time mixed-traffic wireless networks. *IEEE Transactions on Industrial Electronics*, 48(5):883–890.

Yiming, A. and Eisaka, T. (2005). A switched Ethernet protocol for hard real-time embedded system applications. In *19th International Conference on Advanced Information Networking and Applications (AINA 2005)*, volume 2, pages 41–44, Taipei, Taiwan.

Zhang, W. (2001). *Stability analysis of networked control systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, Ohio, USA.

Zhang, W., Branicky, M. S., and Phillips, S. M. (2001). Stability of networked control systems. *IEEE Control Systems Magazine*, 21:84–99.

Zheng, H. and Boyce, J. (2001). An improved UDP protocol for video transmission over Internet-to-wireless networks. *IEEE Transactions on Multimedia*, 3(3):356–365.

Zhou, Y., Ananda, A., and Jacob, L. (2003). A QoS enabled MAC protocol for multi-hop ad hoc wireless networks. In *Proceedings of IEEE International Performance, Computing, and Communications Conference (IPCCC2003)*, pages 149–156, Phoenix, Arizona, USA.