ABSTRACT

HYUN, SANGWON. Secure and Reliable Code Dissemination for Wireless Sensor Networks. (Under the direction of Dr. Peng Ning.)

Wireless sensor networks are considered ideal candidates for a wide range of applications, such as industry monitoring, data acquisition in hazardous environments, and military operations. It is desirable and sometimes necessary to reprogram sensor nodes through wireless links after they are deployed, due to, for example, the need of removing bugs and adding new functionalities. The process of propagating a new code image to the nodes in a wireless sensor network is commonly referred to as code dissemination. Code dissemination requires reliable delivery of code images through lossy wireless channels, typical in wireless sensor networks. In addition, code dissemination must be protected from various malicious attacks to ensure both integrity and availability of the remote programming service. This thesis includes three studies for efficient, secure, and reliable code dissemination in wireless sensor networks.

First, we present the design, development, and evaluation of a secure and DoS-resistant code dissemination system named Seluge for wireless sensor networks. Seluge is a secure extension to Deluge, a de facto code dissemination system for wireless sensor networks. It provides security protections for code dissemination, including the integrity protection of code images and resistance to various DoS attacks that exploit code dissemination protocols. Seluge is superior to all previous attempts for secure code dissemination, and is the only solution that seamlessly integrates the security mechanisms and the Deluge efficient propagation strategies. Besides the theoretical analysis that demonstrates the security and performance of Seluge, we also report the experimental evaluation of Seluge in a network of MicaZ motes, which shows the efficiency of Seluge in practice.

Second, we present a channel migration scheme, which can be used to mitigate wireless jamming attacks against code dissemination in wireless sensor networks. By exploiting the multiple wireless channels typically available on the existing wireless sensor platforms, our scheme uses a flexible and resilient approach to switch communication channels, which enables sensor nodes to continue the normal code dissemination operations in the presence of jamming attacks. A nice property of the proposed scheme is that it does not depend on any single, fixed channel and executes in a decentralized and independent way on each sensor node. To enhance the jamming resilience of Seluge and investigate the effectiveness of the proposed channel migration scheme, we apply the channel migration scheme to Seluge, and evaluate the resulting protocol through both theoretical analysis and experimental evaluation in a testbed of MicaZ motes. Both results indicate that our channel migration scheme can effectively and efficiently mitigate jamming attacks.
Third, we present the design, implementation, and evaluation of an efficient, reliable, and secure code dissemination scheme called *FEC-Seluge*, which integrates *forward error correction (FEC)* into Seluge to improve the efficiency while preserving the same level of security protection as in Seluge. At the heart of FEC-Seluge is a novel proactive transmission mechanism, which achieves both efficiency and loss-tolerance in code dissemination. To balance the efficiency and loss-tolerance capabilities, we develop a technique to estimate the optimal amount of redundancy in proactive transmission for a given network condition. We evaluate FEC-Seluge through experiments in a testbed environment. The experimental results show that FEC-Seluge achieves much better performance than both Seluge and LR-Seluge, an existing FEC extension to Seluge.
Secure and Reliable Code Dissemination for Wireless Sensor Networks

by

Sangwon Hyun

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2011

APPROVED BY:

Dr. Huaiyu Dai

Dr. Douglas S. Reeves

Dr. Ting Yu

Dr. Peng Ning
Chair of Advisory Committee
DEDICATION

I would like to gratefully dedicate this Doctoral Dissertation to my loving wife Dr. Jungsoon Choi, who always gave me heartful encouragement and devoted support by my side during the entire PhD study. Without the invaluable encouragement and support of my wife, it would be impossible to finish this work. I am also grateful to my grandmother Samhee Lee and my parent Dr. Jongmyung Hyun and Younghee Kim for always supporting me in any mean.

Their supports are invaluable.
BIOGRAPHY

Sangwon Hyun is a Ph.D. student in the Department of Computer Science at North Carolina State University from 2005 to now. He received his B.S. degree from the Department of Electrical and Computer Engineering of Sungkyunkwan University in 2002, and received his M.S. degree from the Department of Electrical Engineering and Computer Science of Seoul National University in 2004. His research focuses on developing systems and techniques to enhance the security of networked and distributed systems, specifically the security of wireless sensor networks.
ACKNOWLEDGEMENTS

First, I am grateful for my advisor Dr. Peng Ning, who has provided me with invaluable guidance and suggestions of my research. I would like to thank him for his generous support and encouragement during my Ph.D. study. I would like to thank my committee members, alphabetically, Dr. Huaiyu Dai, Dr. Douglas S. Reeves, and Dr. Ting Yu for their valuable feedback and comments on my dissertation.

I would like to thank my officemates for their supports during my Ph.D. study: Ahmed Moneeb Azab, Jason Gionta, Dr. Chongkyung Kil, Dr. Mihui Kim, Dr. An Liu, Yao Liu, Arun Natarajan, Young-Hyun Oh, Dr. Younghee Park, Barry Peddycord III, Wenbo Shen, Dr. Kyuyong Shin, Jungki So, Dr. Kun Sun, Ruowen Wang, Dr. Attila Altay Yavuz, Xianqing Yu, and Wu Zhou.

Finally, I would like to give my special thanks to my family for their endless encouragement and support.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
<tr>
<td>Chapter 1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Integrity Protection of Code Images and Resistance to DoS Attacks Exploiting Dissemination Protocols</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2 Jamming-Resistant Code Dissemination</td>
<td>4</td>
</tr>
<tr>
<td>1.1.3 Loss-Tolerant and Secure Code Dissemination using Erasure Codes</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Summary of Contributions</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Organization of Thesis</td>
<td>6</td>
</tr>
<tr>
<td>Chapter 2 Background</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Code Dissemination Protocols for Wireless Sensor Networks</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1 Approaches based on ARQ</td>
<td>8</td>
</tr>
<tr>
<td>2.1.2 Approaches based on the Hybrid of Erasure Codes and ARQ</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Anti-jamming Techniques</td>
<td>14</td>
</tr>
<tr>
<td>2.2.1 Spread Spectrum Anti-jamming Techniques</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2 Channel Hopping-based Anti-jamming Techniques</td>
<td>17</td>
</tr>
<tr>
<td>Chapter 3 Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Assumptions and Threat Model</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Design of Seluge</td>
<td>21</td>
</tr>
<tr>
<td>3.2.1 Immediate Authentication of Code Dissemination Packets</td>
<td>21</td>
</tr>
<tr>
<td>3.2.2 Authentication of Page Advertisement and SNACK Packets</td>
<td>25</td>
</tr>
<tr>
<td>3.2.3 Mitigating DoS Attacks against Signature Packets</td>
<td>25</td>
</tr>
<tr>
<td>3.3 Analysis</td>
<td>27</td>
</tr>
<tr>
<td>3.3.1 Security Analysis</td>
<td>27</td>
</tr>
<tr>
<td>3.3.2 Performance Analysis</td>
<td>28</td>
</tr>
<tr>
<td>3.3.3 Comparison with Previous Approaches</td>
<td>30</td>
</tr>
<tr>
<td>3.4 Implementation and Experiments</td>
<td>31</td>
</tr>
<tr>
<td>3.4.1 Implementation</td>
<td>31</td>
</tr>
<tr>
<td>3.4.2 Experimental Evaluation</td>
<td>32</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>37</td>
</tr>
<tr>
<td>Chapter 4 Mitigating Wireless Jamming Attacks via Channel Migration</td>
<td>38</td>
</tr>
<tr>
<td>4.1 Assumptions and Jamming Model</td>
<td>39</td>
</tr>
<tr>
<td>4.2 Mitigating Wireless Jamming Attacks via Channel Migration</td>
<td>40</td>
</tr>
<tr>
<td>4.2.1 Overview of Channel Migration</td>
<td>40</td>
</tr>
<tr>
<td>4.2.2 Channel Quality Estimation</td>
<td>41</td>
</tr>
</tbody>
</table>
# Chapter 5 FEC-Seluge: Efficient, Reliable, and Secure Code Dissemination for Wireless Sensor Networks Using Erasure Codes

5.1 Assumptions and Threat Model .................................................. 58

5.2 Preliminaries ................................................................. 59

5.2.1 Erasure Codes ............................................................ 59

5.2.2 LR-Seluge ................................................................. 59

5.3 Limitations of LR-Seluge ...................................................... 60

5.4 Design of FEC-Seluge ......................................................... 61

5.4.1 Encoding and Authenticating a Code Image ............................. 62

5.4.2 Proactive Packet Transmission .......................................... 63

5.4.3 Optimal Redundancy Rate .............................................. 65

5.4.4 Security Analysis ......................................................... 66

5.5 Experimental Evaluation ..................................................... 67

5.5.1 Optimal Redundancy Rate .............................................. 69

5.5.2 Defense against Bogus Reports ....................................... 69

5.5.3 Comparison of FEC-Seluge, LR-Seluge, and Seluge ............... 71

5.6 Summary ................................................................. 73

# Chapter 6 Conclusion and Future Work ..................................... 74

6.1 Contributions ................................................................. 74

6.2 Future Work ................................................................. 76

References ................................................................. 78
LIST OF TABLES

Table 3.1  Code size (bytes) on MicaZ. .................................................. 32

Table 4.1  Completion Time and Communication Overhead of JR-SeluGe (Seluge is unable to disseminate beyond line 2 in Figure 5.4.) ............................ 54

Table 5.1  Estimated Redundancy Rates under extreme high noise attacks. ........ 71
Table 5.2  Estimated Redundancy Rates under random noise attacks. ................. 71
LIST OF FIGURES

Figure 1.1 Illustration of Code Dissemination; Initiated from the base station, the code image (consisting of three logical units) is distributed over the entire network through the available wireless links denoted by the arrow lines. 2

Figure 2.1 Structure of packets and pages in Deluge. 9
Figure 2.2 The main concept for authentication in [31, 17] 10
Figure 2.3 An example Merkle hash tree of the approach in [14] 11

Figure 3.1 Authentication of code images. 22
Figure 3.2 The Merkle hash tree constructed for page 0 when $M = 8$ and $N = 48$. Packet $Pkt_{0,i}$ ($i = 1, \ldots, M$) consists of $V_{0,i}$ and the values in its authentication path. For example, packet $Pkt_{0,1}$ consists of $V_{0,1}, e_2, e_{3-4}$, and $e_{5-8}$. 23
Figure 3.3 One-way key chain for puzzle keys. 26
Figure 3.4 Message specific puzzles. 27
Figure 3.5 The testbed (65 MicaZ motes; 152.5 feet $\times$ 97 feet). 33
Figure 3.6 Propagation delay. (The Berkeley approach does not protect maintenance packets. When this mechanism was disabled in Seluge, the propagation delay was reduced by 30 – 146 seconds.) 34
Figure 3.7 Communication overhead. 36
Figure 3.8 Dissemination progress over time on selected nodes. 37

Figure 4.1 Illustration of our channel migration scheme 45
Figure 4.2 The sensor network testbed (72 MicaZ motes; 152.5 feet $\times$ 97 feet). 52
Figure 4.3 Completion time and communication overheads of JR-Seluge and Seluge in scenario 1 53
Figure 4.4 Completion time and communication overheads of JR-Seluge in scenarios 3 and 4 55

Figure 5.1 An example sequence of rounds in LR-Seluge 61
Figure 5.2 Encoding and authenticating code images 63
Figure 5.3 An example sequence of rounds in our approach 64
Figure 5.4 The sensor network testbed (72 MicaZ motes; 152.5 feet $\times$ 97 feet). 68
Figure 5.5 FEC-Seluge performance w.r.t. different redundancy rates 70
Figure 5.6 Dissemination delay and number packet transmissions in Seluge, LR-Seluge and its variation, and FEC-Seluge. 72
Chapter 1

Introduction

A wireless sensor network is expected to consist of a potentially large number of low-cost, low-power, and multi-functional sensor nodes that communicate over short distances through wireless links [6]. Due to their potential to provide fine-grained sensing and actuation at a reasonable cost, wireless sensor networks are considered ideal candidates for a wide range of applications, such as industry monitoring, data acquisition in hazardous environments, and military operations.

It may be necessary to update the code image every sensor node is running after they are deployed, due to, for example, the need of removing bugs and adding new functionalities. The large-scale and embedded nature of wireless sensor networks makes it infeasible to physically access them one-by-one to update their code images. An efficient way is to propagate a new code update using the ad-hoc wireless network formed by the sensor nodes. The process of disseminating a new code image to the network nodes in this way is commonly referred to as code dissemination. Figure 1.1 shows a simple illustration of code dissemination. There exist a few code dissemination protocols [13, 24, 57, 30, 50, 29, 52, 44]. In particular, Deluge [24] uses an epidemic protocol [33] for efficient advertisement of code meta data, and divides each code image into multiple pages to allow simultaneous propagation of multiple parts of the image in different regions of the network. Deluge is generally accepted as a de facto code dissemination system for wireless sensor networks as being included in the current TinyOS distributions [5].

1.1 Motivation

In hostile environments, where there may be malicious attacks against wireless sensor networks, code dissemination may face various threats as follows. For example, the adversary may modify or replace the code image being propagated to sensor nodes, introducing malicious code into the network. As another example, the adversary may inject bogus packets to launch Denial-
Figure 1.1: Illustration of Code Dissemination; Initiated from the base station, the code image (consisting of three logical units) is distributed over the entire network through the available wireless links denoted by the arrow lines.

of-Service (DoS) attacks and interfere with the code dissemination process. In addition, the adversary may jam the communication channel for code dissemination for disrupting the normal code dissemination process. Moreover, due to the open and unattended nature of wireless sensor networks, all such attacks may be launched in the form of so called insider attacks where the adversary captures legitimate nodes and misuse them and the sensitive information (e.g., authentication keys) on them for her attacks. Therefore, code dissemination must be protected from such various malicious attacks to ensure both integrity and availability of the remote programming service.

Code dissemination essentially requires reliable delivery of code images, which means that every node in a network must receive a complete new code image without any missing parts. However, due to the hardware limitation and various environmental factors, sensor nodes frequently suffer from unreliability of the wireless links. In addition, code dissemination leverages the effectiveness of overhearing in wireless communication for efficient dissemination; broadcasting a code dissemination packet benefits every node in the communication range who desires the packet. Consequently, a code dissemination protocol must have an efficient way to achieve the reliability requirement in lossy broadcast environments.

1.1.1 Integrity Protection of Code Images and Resistance to DoS Attacks Exploiting Dissemination Protocols

Although the first generation of code dissemination protocols [13, 24, 57, 30, 50, 29, 52, 44] mentioned above accomplished the efficient dissemination of code images using the ad-hoc wireless network, none of them took security into consideration. Thus all of them are vulnerable
Several recent works have attempted to provide secure code dissemination for wireless sensor networks [17, 14, 31]. All these approaches are extensions to Deluge [24]. Lanigan et al. proposed a protocol named Sluice [31] to integrate signature and cryptographic hash functions to provide efficient authentication for code dissemination. This approach follows Deluge to divide each code image into pages. The hash image of each page is included in the previous page, while the hash image of the first page is signed and included in a signature packet. This approach, however, is vulnerable to Denial of Service (DoS) attacks. This is because in Sluice, a node can only perform authentication when an entire page is received, and thus it cannot authenticate a packet immediately after it is received. To exploit this property, the adversary may send a large number of bogus packets during code dissemination; a sensor node, upon receiving a packet, cannot tell if it is authentic or not, and has to save it if possible. As a result, the adversary can force the sensor nodes to save bogus packets but drop some authentic ones.

A scheme similar to Sluice was independently proposed in [17]. In this approach, the hash image of each code dissemination packet is included in the previous packet, and the hash image of the first packet is signed and included in an advertisement packet. A code dissemination packet can be authenticated only when the packet immediately before it has been received and authenticated. The authors of [17] proposed to optimistically store out-of-order packets. Unfortunately, this decision opens a door to the same DoS attacks, since the adversary can again send a lot of bogus packets to exhaust receivers’ buffers. Not storing out-of-order packets is certainly an option to avoid the DoS attacks. However, this will lead to inefficient code dissemination: Losing one packet at a node will result in the retransmission of all the later packets in the same page.

Deng et al. proposed a scheme to improve the DoS-resilience of secure code dissemination by using Merkle hash tree [14]. Besides having the hash image of each page in the previous one, it also uses a Merkle hash tree to allow each packet to be immediately authenticated upon receipt. However, this approach adds additional overhead due to the transmission of a Merkle hash tree for every page [14]. Moreover, it distributes each hash tree in a level-by-level fashion; only after every packet in one level is received and verified can the packets in the next level be requested. This indeed disables the efficient page-by-page propagation in Deluge, leading to higher propagation delays.

Though all existing approaches [17, 31, 14] are based on Deluge [24], none of them provide a satisfactory solution to the authentication of advertisement and Selective Negative Acknowledgment (SNACK) packets. (In Deluge, such packets are used to advertise new data and facilitate the request and retransmission of packets.) Indeed, the approaches in [17, 31] overlooked the authentication of such packets. As a result, the adversary can forge such packets and exploit the Deluge epidemic and suppression mechanisms to launch DoS attacks. For example, the
adversary may repeatedly request packets from a node to exhaust its battery power. Moreover, the adversary may request packets from a non-existing node; due to the Deluge suppression mechanism, others that overhear this request will not send request for packets (to real nodes). The authors of [14] discussed the authentication of SNACK packets, but overlooked the issue of advertisement packets.

Another critical issue common to these approaches [17, 31, 14] is the vulnerability to DoS attacks against the signatures used to bootstrap secure code dissemination. The approaches in [17, 31] did not consider such threats at all. As a result, the adversary can broadcast packets with bogus signatures, and force all the receivers to perform expensive signature verifications. The approach in [14] proposes to release previously undisclosed values in a one-way hash chain to mitigate such DoS attacks. However, this method is vulnerable to online attackers. Once the adversary overhears a hash value during a legitimate code dissemination, it can reuse the value to send forged signatures to other regions of the network. Due to the multi-hop and low-bandwidth nature of wireless sensor networks, the adversary has sufficient time to launch DoS attacks against many sensor nodes.

In Chapter 3, we present the design, development, and evaluation of a secure and DoS-resistant code dissemination system named Seluge for wireless sensor networks, which is able to address all the limitations existing in the previous approaches [17, 31, 14].

1.1.2 Jamming-Resistant Code Dissemination

Another serious threat to the availability of the code dissemination service is wireless jamming attacks. Wireless sensor networks are vulnerable to jamming attacks due to the use of shared wireless medium. A jammer can simply take advantage of a radio frequency (RF) device to transmit interference signals on wireless channels. As a result, signals of the jammer and the sender collide at the receiver, and the legitimate communication is disrupted. Without proper defense against jamming attacks, it is impossible for wireless sensor networks to perform normal code dissemination operations. More specifically, sensor nodes under jamming cannot download a new code update, and consequently cannot run the operations intended with the new code update. Therefore, jamming resistance is crucial for the success of the remote programming service.

Traditional defense against jamming attacks relies on physical layer spread spectrum techniques such as DSSS and FHSS [54, 64]. Several groups of researchers also developed channel hopping approaches for platforms without built-in FH technology [45, 67, 20, 66], in which communicating parties follow the same channel hopping sequence in a coordinated fashion. All those approaches require that the sender and its receivers share a common secret so that they can use the same spreading code or channel hopping sequence for communication. Unfortu-
nately, this feature makes all of them vulnerable to insider jammers, who may learn the shared secret through compromised receivers and then jam the wireless communication.

To address this problem, several research groups recently developed DSSS and FHSS-based anti-jamming broadcast communication schemes by eliminating the requirement of shared secrets in traditional DSSS and FHSS (e.g., [59, 60, 55, 51, 39]). However, these approaches all require changes to the physical radio devices, which are not available on existing wireless sensor platforms.

Lazos et al. proposed a semi-random channel hopping strategy [32] to allow nodes in a cluster to intermittently meet on the same channel to mitigate control-channel jamming attacks. However, this approach can only use a small fraction of the bandwidth due to the need to hide the common channel. In addition, it uses a single cluster header to generate the channel hopping sequences for all cluster members and transmits them to each member individually. Thus, this approach also has a single point of failure and limited scalability.

As discussed above, the existing approaches are either vulnerable to insider jammers, require hardware modification, or degrade the performance. In Chapter 4, we investigate a new anti-jamming solution which does not require any shared secret and hardware modification while allowing better utilization of the available bandwidth.

1.1.3 Loss-Tolerant and Secure Code Dissemination using Erasure Codes

Despite the security protections provided by Seluge and other secure code dissemination schemes [17, 14, 31], all of them rely on ARQ (Automatic Repeat reQuest) for reliable dissemination of code images, which is employed by Deluge. However, this ARQ mechanism is inefficient in lossy broadcast environments, which are typical in code dissemination, due to asymmetric packet loss between neighbor nodes.

Erasure codes – a forward error correction (FEC) technique – are a well-known approach for reliable communication. Several researchers demonstrated recently that using erasure codes along with ARQ can provide efficient and reliable code dissemination (e.g., [22, 21, 53]). Based on these techniques, Bohli et al. integrated both integrity protection and erasure coding into code dissemination [7]. However, this approach does not provide immediate authentication of erasure-encoded packets, and consequently is vulnerable to DoS attacks that exploit authentication delays.

To address the lack of immediate authentication in erasure coding based code dissemination, LR-Seluge [69] was recently developed to securely integrate erasure codes into Seluge [26]. LR-Seluge offers better properties than [7]. Unfortunately, it does not take full advantage of erasure codes due to the lack of proactive transmission. As a result, it does not effectively reduce dissemination delays. In addition, the form of encoding and authenticating code images also
prevents flexible application of erasure codes in LR-Seluge.

In Chapter 5, we investigate several new techniques to address the above problems and develop an efficient, reliable, and secure code dissemination scheme called \textit{FEC-Seluge}.

\section*{1.2 Summary of Contributions}

The contributions of this thesis are summarized below:

1. Seluge: First, we develop the Seluge suite of techniques for secure and DoS-resistant code dissemination in wireless sensor networks. Our security and performance analysis demonstrates that Seluge is superior to all the previous solutions \cite{17, 31, 14}. Second, we integrate the proposed techniques into the Deluge code base in TinyOS distributions, providing a software packet that is readily available. Third, we perform extensive experiments in a network of MicaZ motes to evaluate the performance of Seluge.

2. Channel Migration: The first and most important contribution is the development of the channel migration scheme, which is key to achieving efficient and strong jamming resilience properties. Second, we apply our channel migration scheme to Seluge in a case study and carefully evaluate it through both theoretical analysis and experimental evaluation, which demonstrate the nice properties and real-world performance of our channel migration scheme.

3. FEC-Seluge: First, we develop an efficient, reliable, and secure code dissemination system named FEC-Seluge which seamlessly integrates erasure codes into the security mechanisms of Seluge. At the heart of FEC-Seluge is a novel proactive transmission mechanism, which achieves both efficiency and loss-tolerance through proactively transmitting redundant packets to enable receivers to tolerate packet losses. To balance the efficiency and loss-tolerance capabilities, we also developed a technique to estimate the optimal amount of redundancy in proactive transmission for a given network condition. Finally, we implement FEC-Seluge as an extension to Seluge in TinyOS and perform extensive empirical studies in a network of MicaZ motes to see the performance of FEC-Seluge.

\section*{1.3 Organization of Thesis}

The rest of this thesis is organized as follows. Chapter 2 gives the background information on code dissemination in wireless sensor networks and anti-jamming communication. Chapter 3 presents the design, development, and evaluation of Seluge, a secure and DoS-resistant code
dissemination system for wireless sensor networks. Chapter 4 presents the design, implementation, and evaluation of our channel migration scheme to mitigate wireless jamming attacks. Chapter 5 presents the design, implementation, and evaluation of FEC-Seluge, which seamlessly integrates erasure codes into the security mechanisms of Seluge for efficient, reliable, and secure code dissemination in wireless sensor networks. Chapter 6 concludes this thesis and discusses the future work.
Chapter 2

Background

In this chapter, we give the background information related to our research in this thesis. In Section 2.1, we briefly review the existing code dissemination protocols for wireless sensor networks and point out their limitations. We then discuss the existing techniques to cope with wireless jamming attacks in Section 2.2.

2.1 Code Dissemination Protocols for Wireless Sensor Networks

All the existing code dissemination protocols can be classified into two categories in terms of their mechanisms to achieve reliable dissemination of code images. The first category of protocols (e.g., [13, 24, 57, 30, 52, 44, 14, 17, 31, 26]) is solely based on ARQ (Automatic Repeat Request), in which each node repeatedly requests missing packets until it completely receives the entire code image. We discuss about this category of protocols in Section 2.1.1. The second category of protocols (e.g., [22, 21, 53, 7, 69]) utilizes forward error correction along with ARQ to address the inefficiency problem of ARQ in lossy broadcast environments where asymmetric packet loss between neighbors happens. In Section 2.1.2, we review this category of protocols.

2.1.1 Approaches based on ARQ

Several code dissemination protocols [13, 24, 57, 30, 50, 29, 52, 44] were developed to propagate new code images using the ad-hoc wireless network formed by the sensor nodes. In particular, Deluge [24] is the most widely used as a de facto code dissemination system in TinyOS [5], an open-source operating system for wireless sensor networks. Deluge uses a page-by-page dissemination strategy. A code image is divided into fixed-size pages, and each page is further split into same-size packets. Figure 2.1 shows the construction of packets and pages. The pages for a code image is delivered in a sequential order. After completely receiving a page, a
node advertises the availability of the newly received page, and may transmit the corresponding packets upon request. On the other hand, a receiver requests a page only after having received all packets in previous pages.

Deluge uses an epidemic protocol [33] for efficient advertisement of code meta data [24]. Each node periodically advertises the version of its code image and the number of pages it has for that version. For energy efficiency, the advertisement rate is dynamically adjusted: If a node discovers its own advertisement is different from those received from others, it increases its advertisement rate. Otherwise, it decreases the rate. As a result, Deluge can achieve rapid propagation during dissemination of a new code image, but consumes little resource in steady state.

Once a node finds out from an advertisement packet that a neighbor node has the page it needs, it uses Selective Negative Acknowledgment (SNACK) packets to request transmission from this neighbor. Each SNACK packet contains a requested page number and a bit vector indicating the requested packets. Upon receiving several request packets for the same page, a node computes the union of the requested packets, and transmits the packets in a round-robin fashion.

Deluge uses various message suppression mechanisms for efficiency reasons. To reduce redundant advertisements, each node suppresses its own advertisement if the number of overheard advertisement packets having the same information is over a predefined threshold. Moreover, if a node overhears request (or data) packets for a page that it is about to request or has already received, it suppresses its own request packet. Similarly, if a node overhears request (or data) packets for the pages with smaller indices than that of the page it is currently transmitting, the node suppresses transmission of the subsequent data packets. By using such suppression mechanisms, Deluge increases the possibility that nodes in the same region wait for the same page and consequently maximizes the effect of overhearing.
However, Deluge and the other protocols [13, 57, 30, 50, 29, 52, 44] mentioned above never take into account security issues they may face in hostile environments. Consequently, the adversary may simply misuse these protocols to distribute her own malicious code images to the network, and launch DoS attacks for disrupting the normal operations of these protocols.

**Secure Extensions to Deluge**

Several groups of researchers attempted to provide security protections to Deluge [31, 17, 14, 62]. Lanigan et al. proposed a protocol named Sluice [31] to integrate signature and cryptographic hash functions to provide efficient authentication for code dissemination. In Sluice [31], authentication is done in page granularity. Figure 2.2 illustrates how Sluice authenticates a given code image. Initially, the hash image of the last page\((Page_n)\) is computed and appended to \(Page_{n-1}\). Then the hash image of \(Page_{n-1}\) including \(h_n\) is computed and appended to \(Page_{n-2}\). Sluice repeats such procedure until getting \(h_1\). Finally, \(h_1\) is digitally signed and included in the signature packet along with the digital signature. When receiving a code image, the sensor node first requests the signature packet and verifies \(h_1\) in it through a signature verification operation. Then it requests \(Page_1\) and verifies it by comparing the hash image computed from the received \(Page_1\) with \(h_1\) already verified. In such way, the node can verify all \(n\) pages. In Sluice, each page can be verified only after receiving and verifying the signature packet and all the previous pages. Because of the authentication in a page unit, an individual code dissemination packet cannot be immediately verified upon receipt, and this property makes Sluice vulnerable to DoS attacks exploiting authentication delays.

The approach in [17] is similar to Sluice except that the granularity of the authentication is packet rather than page. The hash image of each code dissemination packet is included in the previous packet as illustrated in Figure 2.2, and the hash image of the first packet is signed. Therefore, in this approach each packet can be verified only after receiving and verifying all the previous packets including the signature packet. However, due to the possibility of out-of-order delivery of the packets in a same page, this approach does not guarantee immediate authentication of each code dissemination packet.

The approach of [14] was proposed to achieve immediate authentication of code dissemination packets by using Merkle hash tree [42]. This approach constructs a single Merkle hash tree for each page. Figure 2.3 shows a Merkle hash tree when each page consists of 9 packets and each packet payload can afford up to 3 hash images. Each node of the tree corresponds to

![Figure 2.2: The main concept for authentication in [31, 17]](image-url)
a single packet. Each code dissemination packet ($Pkt_4$ to $Pkt_9$) in a page becomes a leaf node of the tree. An internal node contains the hash images of all its child nodes, and the number of the children of each internal node depends on the number of hash images the corresponding packet payload can include. The hash image of each root packet ($Pkt_1$) is included in the root packet of the hash tree for the previous page, and the hash image of the root packet of the first page is signed. In order to verify a packet, each node must have received and verified the signature packet and all the packets in the previous pages. In addition, the node must have received and verified all the packets on the path from the root to the parent of the packet in the tree, for instance, $Pkt_1$ and $Pkt_2$ to verify $Pkt_5$. To achieve immediate authentication, the authors proposed the level-by-level transmission of the packets of each hash tree; only after every packet in one level is received and verified can the packets in the next level be requested. Unfortunately, this indeed disables the efficient page-by-page propagation in Deluge, leading to higher propagation delays.

Tan et al. proposed a secure code dissemination approach [62] solely based on symmetric key cryptographic operations. This approach uses multiple one-way key chains, each of which is used to authenticate code dissemination packets destined to a group of nodes with the same hop distance from the base station. During the initialization phase, the base station generates the one-way key chains for every group of nodes, and then securely distributes each key chain commitment to the corresponding group of nodes. Given a new code image to disseminate, the base station authenticates the $i$th code dissemination packet of the new code image with the $i$th element $k_i$ of each key chain, and then broadcasts the authenticated packet along with the encrypted $k_i$ with $k_{i-1}$. The receiving node first retrieves $k_i$ through a decryption operation...
and then verifies the packet with it. However, in this approach, if a compromised node has received a packet with $k_i$ while some of the neighbors has missed the packet, it is possible for the compromised node to provide forged packets to those neighbors before the neighbors receive the legitimate packet with $k_i$. In addition, this approach does not achieve immediate authentication in case of out-of-order packet delivery. In addition, the management of the key chains requires additional overhead.

Though all the above-mentioned approaches [17, 31, 14, 62] are based on Deluge, none of them provide a satisfactory solution to the authentication of advertisement and Selective Negative Acknowledgment (SNACK) packets. Indeed, the approaches in [17, 31, 62] overlooked the authentication of such packets. As a result, the adversary can forge such packets and exploit the Deluge epidemic and suppression mechanisms to launch DoS attacks. For example, the adversary may repeatedly request packets from a node to exhaust its battery power. Moreover, the adversary may request packets from a non-existing node; due to the Deluge suppression mechanism, others that overhear this request will not send request for packets (to real nodes). The authors of [14] discussed the authentication of SNACK packets, but overlooked the issue of advertisement packets.

Another critical issue common to the approaches in [17, 31, 14] is the vulnerability to DoS attacks against the signatures used to bootstrap secure code dissemination. The approaches in [17, 31] did not consider such threats at all. As a result, the adversary can broadcast packets with bogus signatures, and force all the receivers to perform expensive signature verifications. The approach in [14] proposes to release previously undisclosed values in a one-way hash chain to mitigate such DoS attacks. However, this method is vulnerable to online attackers. Once the adversary overhears a hash value during a legitimate code dissemination, it can reuse the value to send forged signatures to other regions of the network. Due to the multi-hop and low-bandwidth nature of wireless sensor networks, the adversary has sufficient time to launch DoS attacks against many sensor nodes. In Seluge we present in Chapter 3, we address all the above-mentioned limitations existing in [17, 31, 14, 62].

Besides of all these approaches, there are a few more approaches for secure code dissemination for wireless sensor networks. In [63], Tan et al. proposed an approach for efficiently protecting the confidentiality of code images. In order to preserve the confidentiality, each code dissemination packet is encrypted with its own hash image as the encryption key, and the hash image is distributed as an attachment to the previous packet. In addition, they integrated mechanisms to deal with signature-based and SNACK-based DoS attacks, which are similar to Seluge. Most recently, Ugus et al. attempted to reduce the memory requirement of the existing secure code dissemination approaches by eliminating their requirements of a public-key cryptography primitive, whose implementation usually requires a large amount of memory [65]. To replace the public-key cryptography primitive, they developed the stateful-verifier $\tau$-time sig-
nature scheme which is built on Merkle’s one-time signature scheme [43]. Since the introduced stateful-verifier τ-time signature scheme only relies on a secure one-way hash function, it is not only computationally efficient but also requires a small memory footprint. Finally, they applied the proposed signature scheme to an existing secure code dissemination approach, resulting in much smaller memory footprint.

2.1.2 Approaches based on the Hybrid of Erasure Codes and ARQ

In order to improve the efficiency of code dissemination in wireless sensor networks, several researchers developed code dissemination protocols which utilize either network coding or erasure coding along with ARQ (e.g., [22, 21, 53]). Hou et al. proposed an approach named Adapcode which uses adaptive network coding together with ARQ for efficient and reliable dissemination of code images in wireless sensor networks [22]. Given the observation that the best coding scheme which enables the best code dissemination performance varies depending on the number of neighbors, in this approach each sensor node dynamically changes the coding scheme based on its current observation on the number of neighbors. In addition, the authors suggested a sender selection mechanism to evenly distribute the number of packet transmissions over different senders.

Hagedorn et al. and Rossi et al. proposed code dissemination protocols based on rateless erasure codes. The approach in [21] employed random linear codes, while the approach in [53] used Fountain codes [40]. The use of erasure codes allows both approaches to significantly reduce retransmission requests and consequently dissemination latency as well as communication overhead. Moreover, due to the rateless property, in both approaches the sender can generate an arbitrarily number of encoded packets depending on its local neighbors’ requests. That is, it is possible to adaptively change the code rate according to the local link quality. Despite such efficiency with either network coding or rateless erasure coding, none of these approaches took security into consideration.

Secure Extensions

In [7], Bohli et al. proposed a technique which prevents illegitimate code images being propagated in the approach of [53]. In this approach, the integrity of each code page is verified after decoding it. However, this approach does not provide immediate authentication of each encoded packet upon receipt, and consequently is vulnerable to DoS attacks that exploit authentication delays. To cope with this problem, the authors suggested a strategy for filtering bogus encoded packets, but did not provide enough discussion on the effectiveness of the strategy.

To address the lack of immediate authentication in erasure coding based code dissemination, LR-Seluge [69] was recently developed to securely integrate erasure codes into Seluge. By
adapting fixed rate erasure codes and properly organizing the hash images of encoded packets, LR-Seluge not only provides loss-resilience in code dissemination, but also achieves immediate authentication of every encoded packet. In addition, to optimize the way to use encoded packets, LR-Seluge suggests a packet transmission strategy which selects the packet desired by more receivers with higher priority and eventually broadcast the minimum set of packets that can satisfy all the receivers’ needs of additional packets required for decoding. However, this strategy prevents LR-Seluge from taking full advantage of erasure codes due to the lack of proactive transmission, and consequently LR-Seluge does not effectively reduce dissemination delays.

2.2 Anti-jamming Techniques

There are many existing techniques to cope with wireless jamming attacks. Physical layer spread spectrum techniques such as DSSS and FHSS are traditional defense against wireless jamming attacks [54, 64]. Several new anti-jamming schemes based on DSSS and FHSS were recently developed to overcome the limitations in those traditional spread spectrum techniques [59, 60, 55, 51, 39, 35, 36]. In Section 2.2.1, we briefly review the basic ideas of DSSS and FHSS, and then discuss recently developed anti-jamming techniques based on DSSS and FHSS. Several groups of researchers also developed anti-jamming approaches using upper layer channel hopping for platforms without built-in frequency hopping technology [45, 67, 20, 66, 32]. We discuss these approaches in Section 2.2.2.

2.2.1 Spread Spectrum Anti-jamming Techniques

DSSS-based Anti-jamming Techniques

Direct Sequence Spread Spectrum (DSSS) is a traditional anti-jamming technique. In DSSS communication, the sender spreads a given message into a bit stream called chips by multiplying each bit of the message with a spreading code, which is a sequence of bits. The sender transforms the spread result into an RF signal and then transmits the signal on the wireless channel. Upon receiving the RF signal, the receiver first recovers the chips from the signal. Then the receiver despreads the recovered chips to get the original message using the same spreading code used by the sender.

Spreading a message in DSSS makes the signal bandwidth much wider than the bandwidth needed to transmit the information in the message. Despreading distributes the energy of narrow-band jamming signals to the entire bandwidth, and consequently enables the receiver to successfully recover the original message in the presence of the jamming signals. However, DSSS requires that the sender and receivers share a common secret for generating the same
spreading code required for them to establish anti-jamming communication. Unfortunately, this feature makes DSSS vulnerable to insider jammers, who may learn the shared secret through compromised receivers and then jam the wireless communication.

Pöpper et al. recently developed Uncoordinated DSSS [51], which enables jamming-resistant broadcast communication by eliminating the requirement of shared secrets in DSSS. In UDSSS, both sender and receivers share the same set of spreading codes. The spreading codes in the set are grouped into code sequences. Given a message to transmit, the sender randomly chooses a code sequence from the set, and spreads each bit of the message using a spreading code in the chosen code sequence. Such random choice makes jammers hard to quickly find out the code sequence used by the sender. However, to despread the message, the receiver also needs to try each code sequence in the set one-by-one until finding the correct code sequence used by the sender. This procedure introduces high computational overhead on the receiver side. In addition, if the jammer has sufficient computational power so that she can find the code sequence used by the sender before the sender finishes transmission, the jammer can jam the remaining part of the transmission. Thus, UDSSS is vulnerable to reactive jammers with sufficient computational power.

To address the vulnerability to reactive jammers in UDSSS, Liu et al. proposed a scheme named RD-DSSS [39] which uses multiple code sequences and permutation to protect each message. However, this approach not only increases burdens on the jammer side, but also requires more efforts on the receiver to find the code sequence randomly chosen by the sender. Thus, the same high computational overhead problem still remains in this approach.

Most recently, Liu et al. proposed DSD-DSSS for efficient anti-jamming broadcast communication [36]. In this approach, the sender spreads each message using the code sequence randomly generated based on a random seed only known to the sender. In order to achieve anti-jamming capability, the sender delays the timing when it discloses the random seed after the message transmission has been completed. In this way, when the jammer has the knowledge of the random seed and code sequence which is required for jamming the message transmission, all the receivers have already received the message. A possible jamming strategy against DSD-DSSS is to jam the seed disclosure. To prevent this type of jamming attacks, the authors developed a content-based code subset selection scheme which allows a normal receiver who has received the full message including the random seed to quickly decode the random seed. On the other hand, a jammer without knowing the complete message has more choices to consider.

**FHSS-based Anti-jamming Techniques**

Frequency Hopping Spread Spectrum (FHSS) is another type of spread spectrum technique for anti-jamming. In FHSS, the sender and receiver first generate the same frequency hopping
sequence using the shared secret between them. The sender and receiver communicate with each other, switching their frequencies according to the same hopping sequence in a synchronized way. As long as the jammer has no knowledge of the hopping sequence, the probability that the jammer hits the same channel as the sender and receiver is very low if the number of all available channels is large enough. However, due to the requirement of a pre-shared secret between the sender and its receivers, FHSS is also vulnerable to insider jammers as in DSSS.

Strasser et al. recently proposed UFH (Uncoordinated Frequency Hopping) message transfer protocol [59], whose objective is for a sender and receiver without any pre-shared secret to establish a shared secret in the presence of jammers. After a shared secret is established between them, they can use DSSS or FHSS to protect their subsequent communication from jamming attacks. The basic idea of UFH message transfer protocol is to have the sender quickly hop through random channels and repeat the same transmission, and have a receiver slowly hop through the same set of channels so that it can receive the transmission when the sender hits its channel. To allow very short hop duration for the sender, UFH message transfer protocol fragments the message into short packets, and the receiver reassembles the received fragments to get the original message. To deal with the problems due to fake fragments from attackers, the authors also proposed hash chaining approach to verify each message fragments.

A few variations of UFH were developed to improve the efficiency of UFH. In [60], Strasser et al. proposed several new UFH communication schemes based on erasure codes and packet verification techniques to build verifiable message coding. They showed that BMA (erasure coding combined with a oneway authenticator based on linear maps) reduces the communication latency of UFH up to one-half. In [55], Slater et al. also developed several schemes (i.e., Hashcluster, Merkleleaf, and Witnesscode schemes) using erasure codes to improve the original UFH protocol.

Unfortunately, all these schemes [59, 60, 55] share a common limitation: They have to split each message into multiple short packets due to the constraint of packet size, which is determined by the hop duration and the bit rate. Due to the need to reassemble these packets into meaningful and authenticated messages, each packet has to include additional fields and thus cannot be shorter than a certain length. This feature makes all these schemes vulnerable to responsive jamming attacks, which sense and jam channels with meaningful signals reactively. As a result, it takes a long time (and sometimes impossible) for these schemes to finish transmitting a meaningful message in the presence of jammers.

Most recently, Liu et al. proposed an anti-jamming scheme named USD-FH [35] to improve resilience against responsive jamming attacks. The key idea in this approach is to transmit each message by following the pseudo-random frequency hopping pattern derived from a seed and disclose the seed in an uncoordinated fashion (on a random frequency band) prior to the message transmission. When there are a large number of non-overlapping frequency bands
available for wireless communication, given the assumption that jammers are unable to jam the entire bandwidth at the same time, there is always a chance that only a normal receiver receives the seed and accordingly the message while a jammer does not. To increase this possibility, USD-FH repeats the transmission of a seed and the corresponding message multiple times.

Besides the techniques discussed above, Chiang et al. proposed a scheme that can work with any existing spread spectrum systems so that the systems can detect and isolate insider jammers [10]. The scheme uses a type of binary tree called code tree, in which each node corresponds to a code used for spread spectrum communication. In this scheme, if some of the codes in the tree are captured and being used for jamming, the transmitter can identify those captured codes by cooperating with its receivers, and then remove them from the system.

2.2.2 Channel Hopping-based Anti-jamming Techniques

Several groups of researchers developed channel hopping approaches for platforms without built-in FH technology (e.g., [67, 45, 20, 66, 32]). First, Xu et al. proposed two different channel surfing approaches named coordinated channel switching and spectral multiplexing [67]. In these approaches, every network node is pre-configured with a common channel sequence and equipped with a jamming detection mechanism. In normal condition where there exists no jamming, every network node resides and communicates on a single, fixed channel. In the coordinated channel switching, if jamming is detected, the entire network switches to the next channel in the pre-configured channel sequence. In the spectral multiplexing, only the nodes inside the jammed region switches to the next channel in the channel sequence. The nodes at the boundary of the jammed region act as radio relays, constantly switching between the original and the new channel where the nodes under the jamming currently reside.

Navada et al. proposed a channel hopping approach to protect 802.11 networks from wireless jamming attacks [45]. To establish anti-jamming communication, first of all, the AP generates a seed value to be used to generate a pseudo-random channel hopping sequence and then distributes the seed to its clients via secure communication. For jamming resilience, the AP and clients change their channels according to the same channel hopping sequence generated from the shared seed every period of time called residence time. For synchronous channel hopping, whenever hopping, the AP broadcasts an end-of-slot message to notify the clients on the same channel of its channel switch. Once overhearing the end-of-slot message, clients hops to the next channel in the hopping sequence.

In [20], Gummadi et al. independently proposed a similar channel hopping technique to [45] to enhance the jamming resilience of 802.11 networks. If the AP detects link degradation, it generates a MD5 seed for generating a pseudo-random channel hopping sequence and starts hopping. Each client losing the connection with the AP due to the AP’s channel hopping
begins scanning all channels to find the AP. While scanning, the client transmits a probe request on its channel. Once hearing the probe request, the AP broadcasts a probe reply, which contains the AP’s current encrypted MD5 value. If the client receives the probe reply, it can generate the same channel hopping sequence as the AP using the received MD5 value, and eventually synchronize with the AP. Then the AP and clients resume their communication while periodically hopping through the common channel hopping sequence.

Wood et al. presented a channel hopping scheme to mitigate jamming attacks in wireless sensor networks [66]. As in FHSS, two communicating parties first agree on the same channel hopping sequence derived from their pre-shared key, and then start hopping. In this approach, the $i$th message is transmitted through the $i$th channel in the hopping sequence.

However, all the above mentioned schemes [45, 67, 20, 66] share a common limitation: They all require that the sender and its receivers share the same channel hopping pattern. Unfortunately, this feature makes all of them vulnerable to insider jammers, who may learn the shared channel hopping sequence through compromised receivers and then jam the wireless communication.

Lazos et al. proposed a semi-random channel hopping strategy [32] to mitigate control-channel jamming attacks. The key idea is to generate a pseudo-random channel hopping pattern for each network node in such a way that the hopping patterns of the nodes in the same network cluster have intermittent overlappings which successfully construct a common control channel over the cluster. However, this approach can only use a small fraction of the bandwidth due to the need to hide the common channel. In addition, it uses a single cluster header to generate the channel hopping sequences for all cluster members and transmits them to each member individually. Thus, this approach also has a single point of failure and limited scalability.
Chapter 3

Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks

In this chapter, we present the design, development, and evaluation of a secure and DoS-resistant code dissemination system named Seluge for wireless sensor networks. Seluge is an extension to Deluge [24], an open source code dissemination system included in TinyOS [5]. Seluge inherits the efficiency and robustness properties from Deluge, and at the same time provides security protections for code dissemination, including the integrity protection of code images and resistance to various DoS attacks. Seluge is targeted at the current and future generation of sensor platforms, such as MicaZ [3], TelosB [4], and Imote2 [2].

The key contribution of Seluge is a novel way to organize the packets used to distribute new code images. By carefully arranging code dissemination data items and their hash images in packets, Seluge provides immediate authentication of each packet when it is received, without disrupting the efficient propagation mechanisms used by Deluge. Thus, it can easily defeat the DoS attacks that exploit possible authentication delays.

Seluge uses a signature to bootstrap the authentication of a new code image. However, unlike the previous attempts, Seluge uses a weak authentication along with the signature. This weak authentication mechanism has some nice properties: It can be efficiently verified by a regular sensor node, but it takes a computationally powerful attacker a substantial amount of time to forge. Moreover, it cannot be pre-computed. Thus, this weak authentication mechanism provides an effective filter of forged signatures. As a result, Seluge is not subject to the same DoS attacks against signature verifications as the previous approaches.

Seluge properly authenticates advertisement and SNACK packets. As a result, it can prevent DoS attacks that exploit the Deluge epidemic propagation and suppression mechanisms. This
is another security property not available in the previous approaches.

Compared with the previous attempts [17, 31, 14], Seluge not only provides integrity protection for disseminated code images, but is also resistant to various DoS attacks exploiting the expensive signature verification operations, possible authentication delays, and the efficient epidemic propagation strategies used by Deluge. Indeed, Seluge is superior to all the previous solutions [17, 31, 14], and is the only solution that seamlessly integrates the security mechanisms and the efficient Deluge propagation strategies.

The organization of this chapter is as follows. The next section clarifies our assumptions and the threats to code dissemination in wireless sensor networks. Section 3.2 presents the techniques used in Seluge for secure and DoS-resistant code dissemination. Section 3.3 provides theoretical analysis of the security and performance of Seluge. Section 3.4 describes the implementation and experimental evaluation of Seluge in a network of MicaZ motes. Section 3.5 summarizes this chapter.

3.1 Assumptions and Threat Model

Assumptions: We assume the source of the code images, i.e., the base station, is a powerful node (e.g., a laptop PC) with sufficient energy supply. We assume that sensor nodes are resource constrained. A sensor node may be able to perform a limited number of public key cryptographic operations. For example, a MicaZ mote can perform a 160-bit ECC signature verification operation in about 2.43 seconds using TinyECC [34]. However, a node cannot afford performing many such operations due to the intensive computation and energy consumption. We assume the packet size is large enough to hold a signature and other information required by a signature packet. This can be satisfied on sensor platforms with IEEE 802.15.4 compliant radios [27], where the maximum payload size is 102 bytes. We assume each node has enough memory to store the disseminated code image (e.g., using the measurement flash).

We assume Deluge as the underlying code dissemination protocol. We assume the base station has a private and public key pair, and each sensor node in the network is pre-configured with the base station’s public key. We also assume sensor nodes are able to establish pairwise keys between neighbor nodes, for example, using one of the existing schemes (e.g., [9, 37, 15].

Threat Model: We assume the adversary has access to computationally resourceful nodes such as laptops. The adversary may launch both external and insider attacks. In external attacks, the adversary does not control any valid node in the network. The adversary may attempt to eavesdrop for sensitive information, inject forged messages, replay previously intercepted messages, and impersonate valid sensor nodes. Moreover, the adversary may fake non-existing links by launching wormhole attacks [23]. The adversary may use Sybil attacks [46], where one node presents multiple identities to defeat typical fault tolerant mechanisms. The adversary may
launch DoS attacks by, for example, forging a large number of signature packets or exploiting weaknesses of the code dissemination protocol. The adversary may jam the communication channel; however, we assume that the adversary cannot constantly jam the communication channel without being detected and removed.

The adversary may compromise some nodes to attack the rest of the network. We call such attacks *insider attacks*, since the compromised nodes are considered a part of the network before they are identified and removed. However, we assume that the majority of the nodes are not compromised. The adversary may exploit the compromised nodes in arbitrary ways to attack the remaining nodes. For example, the adversary may instruct the compromised nodes to intercept sensitive information even if the messages are encrypted, (selectively) drop packets, and launch Sybil attacks [46]. The adversary may also instruct the compromised nodes not to cooperate with others, inject false data, and exploit specific weaknesses of various protocols. However, we assume the base station cannot be compromised.

With those capabilities, the adversary attempts to disseminate illegal code images into the sensor network using the code dissemination mechanism, or launch DoS attacks to consume the limited resources (e.g., battery power, memory) on select sensor nodes.

### 3.2 Design of Seluge

Seluge relies on Deluge [24] for efficiency (via epidemic propagation and suppression) and robustness (via SNACK). To defend against the security threats against code dissemination, Seluge further adds three layers of protection: (1) Immediate authentication of code dissemination packets, (2) authentication of page advertisement and SNACK packets, and (3) anti-DoS protection for signature packets. The key contribution of Seluge is that it provides authentication and DoS-resistant protections by efficiently using cryptographic primitives, and at the same time still allows the efficient code dissemination mechanisms in Deluge.

We use the following notations: $H(\cdot)$ denotes a cryptographic hash function; $\text{Sig}(M)$ denotes the signature of $M$ signed by the base station; $C \parallel D$ denotes the concatenation of $C$ and $D$; and $|E|$ denotes the size of $E$ in byte.

#### 3.2.1 Immediate Authentication of Code Dissemination Packets

Following Deluge [24], we partition the code image to be disseminated into fixed-size pages. (For simplicity, we assume all pages have the same size. Our description can be modified slightly to accommodate the cases where the last page has a smaller size.) Assume there are $P$ pages, denoted as page 1 through page $P$. We split each page $i$ ($1 \leq i \leq P$) into $N$ fixed-size packets, denoted as $Pkt_{i,1}$ through $Pkt_{i,N}$. 
As discussed earlier, due to the Deluge page-by-page dissemination strategy, only after successfully receiving all packets in the current page does a node request the next page from a sender. We exploit this property to enable immediate authentication of each received packet at a node.

**Construction of Code Dissemination Packets:** Figure 3.1 illustrates our authentication scheme for the code image to be disseminated. We append the hash image of each packet in page $P$ to the corresponding packet in page $P - 1$. For example, the hash image of packet $Pkt_{P,1}$, $H(Pkt_{P,1})$, is included in packet $Pkt_{P-1,1}$. We then include the hash image of each packet in page $P - 1$ in the corresponding packet in page $P - 2$. This process continues until we finish hashing all the packets in page 2 and including their hash images in the corresponding packets in page 1.

As shown in Figure 3.1, we use Merkle hash tree [42] to facilitate the authentication of the hash images of the packets in page 1. We refer to the packets related to this Merkle hash tree collectively as page 0. Figure 3.2 illustrates the construction of page 0 and its packets. Specifically, we concatenate the hash images of the page 1 packets to form $HashValues = H(Pkt_{1,1}) \parallel \cdots \parallel H(Pkt_{1,N})$, and then fragment $HashValues$ into $M = 2^k$ pieces, where $k$ is
Figure 3.2: The Merkle hash tree constructed for page 0 when \( M = 8 \) and \( N = 48 \). Packet \( Pkt_{0,i} \) \((i = 1,..., M)\) consists of \( V_{0,i} \) and the values in its authentication path. For example, packet \( Pkt_{0,1} \) consists of \( V_{0,1}, e_2, e_3-4, \) and \( e_5-8 \).

We include the root of the Merkle hash tree, the meta data about the code image (e.g., version number, size), and a signature over all of them in a signature packet. For the sake of the minimum value that satisfies

\[
\frac{N \cdot |H(\cdot)|}{2^k} + k \cdot |H(\cdot)| \leq \text{Maximum payload size.} \tag{3.2.1}
\]

(The condition in Equation 3.2.1 is to make sure each leaf node and its authentication path can be transmitted in one packet.) We denote the resulting fragments as \( V_{0,1}, V_{0,2}, \ldots, V_{0,M}, \) and construct a Merkle hash tree using \( V_{0,1}, V_{0,2}, \ldots, \) and \( V_{0,M} \) as leaf nodes [42]. Figure 3.2 shows the construction of the Merkle hash tree when \( N = 48 \) and \( M = 8 \). Specifically, we compute \( e_i = H(V_{0,i}) \) \((i = 1, 2, \ldots, M)\), and build a binary tree by computing internal nodes from adjacent children nodes. Each internal node is the hash image of the two children nodes. For example, in Figure 3.2, \( e_{1-2} = H(e_1||e_2) \), and \( e_{1-4} = H(e_{1-2}||e_{3-4}) \).

We then construct \( M \) packets using this Merkle hash tree. Specifically, we construct one packet \( Pkt_{0,i} \) for each \( V_{0,i}, \) where \( i = 1,2,\ldots,M; \) each packet \( Pkt_{0,i} \) consists of \( V_{0,i} \) and the values in its authentication path (i.e., the siblings of the nodes in the path from \( V_{0,i} \) to the root) in the Merkle hash tree. For example, packet \( Pkt_{0,1} \) consists of \( V_{0,1}, e_2, e_{3-4}, \) and \( e_{5-8} \) in Figure 3.2.
presentation, we refer to the packets in page 1 through page \( P \) as *data packets*, and the packets in page 0 as *hash packets*. In addition to the payload discussed earlier, each packet also has a header describing auxiliary information about the code image, pages, and packets.

The reader may have noticed that we must choose parameters \( N \) and \( M \) carefully to ensure that the hash packets are small enough to be transmitted in wireless sensor networks, as indicated in Equation 3.2.1. This is indeed well accommodated by the current generation of sensor platforms that use IEEE 802.15.4 compliant radios \([27]\), such as MicaZ \([3]\) and TelosB \([4]\) motes. For example, we may set the number of packets per page \( N = 48 \), following the default configuration in Deluge. We may use the 64-bit truncation of SHA-1 as \( H(x) \), which provides sufficient pre-image resistance and has been used previously (e.g., \([17]\)). Moreover, we may set the number of leaf nodes in the Merkle hash tree \( M = 8 \). As a result, each hash packet consists of 6 hash images of page 1 packets (48 bytes) and 3 hash images (24 bytes) in the authentication path in the Merkle hash tree. The total payload size is 72 bytes, smaller than the 102 bytes maximum payload size in the IEEE 802.15.4 standard \([27]\).

**Transmission & Authentication of Code Dissemination Packets:** We rely on the underlying Deluge protocol to distribute packets for a given code image (in a *page-by-page* fashion). The additional capability provided by our packet construction is the *immediate authentication* of packets received by each node. This property is critical to sensor nodes in order to prevent DoS attacks aimed at exhausting receivers’ buffers. Note that the approaches in \([17, 31]\) do not provide this property, and thus are vulnerable to such DoS attacks. Though not vulnerable to such DoS attacks, the approach in \([14]\) is much less efficient than Deluge.

The base station first broadcasts the signature packet, which serves as the advertisement of the new code image. Upon receiving a signature packet, each node verifies the signature to authenticate the root of the Merkle hash tree constructed for page 0. This root allows the node to authenticate each hash packet in page 0 upon receipt, using the values in the authentication path included in the same packet. For example, in Figure 3.2, if \( e_{1-8} \) has been authenticated in the signature packet, upon receiving a packet consisting of \( V_{0,1}, e_2, e_{3-4}, \) and \( e_{5-8} \), a node can immediately verify whether \( H(H(H(V_{0,1}) || e_2) || e_{3-4}) || e_{5-8} = e_{1-8} \). If yes, the received packet is accepted; otherwise, it must be a forged packet and should be discarded right away.

Since the hash packets include all the hash images of the page 1 packets, successful receipt of them allows the node to further authenticate page 1 packets immediately upon receiving them. To continue the above example, the receipt of an authenticated \( V_{0,1} \), where \( V_{0,1} = H(Pkt_{1,1}) || \cdots || H(Pkt_{1,6}) \), implies the correct receipt of \( H(Pkt_{1,1}), \ldots, H(Pkt_{1,6}) \). Thus, when this node receives packet \( Pkt_{1,i} (i = 1, \ldots, N) \) in page 1, it can immediately verify whether hashing \( Pkt_{1,i} \) results in \( H(Pkt_{1,i}) \) and decide whether the received packet should be accepted or discarded. Following the same reasoning, receipt of the packets in page \( i \) \((i = 0, 1, \ldots, P - 1)\) allows a node to authenticate all the packets in page \( i + 1 \) independently and immediately after
those packets are received.

### 3.2.2 Authentication of Page Advertisement and SNACK Packets

As discussed in Section 1.1.1, several efficient code propagation mechanisms used by Deluge are vulnerable to exploits. The root cause of these vulnerabilities is the lack of authentication. We have provided immediate authentication of code packets in the previous subsection. However, the adversary can still exploit page advertisement and SNACK packets. Thus, advertisement and request packets must be authenticated as well. Because of the heavy use of overhearing and suppression, such authentication must be (local) broadcast authentication, i.e., a node can authenticate any packet transmitted by its neighbors.

We use **cluster keys** for local broadcast authentication. (This was first mentioned in [14]; however, the authors of [14] did not give specific details.) Specifically, each node generates a per-node cluster key, which is intended to authenticate all the advertisement and SNACK packets transmitted from itself. When a node is deployed, it notifies its neighbors through periodic hello packets. We assume sensor nodes can establish pairwise keys between neighbor nodes using an existing scheme (e.g., [9, 37, 15]). Upon receiving a hello packet from a new neighbor, after a random delay, each node replies with its cluster key to the sender encrypted using their pairwise key. Moreover, a node that just sends a cluster key to a new neighbor also broadcasts a hello packet so that the new neighbor can reply with its own cluster key.

For each outgoing page advertisement or SNACK packet, the sender includes a unique sequence number (to prevent replay attacks), and authenticates the packet using its cluster key. Each node stores the cluster keys of its neighbors. For each incoming page advertisement or SNACK packet, a node uses the sender’s cluster key to verify its integrity. A node simply discards unauthenticated or duplicate packets. The limitation of this approach is that it cannot uniquely identify senders. As a result, a compromised node can pretend to be its neighbors using their cluster keys.

In an earlier version [25], we discussed a complementary approach that uses $\mu$TESLA for local broadcast authentication. The $\mu$TESLA approach provides true broadcast authentication of page advertisement and SNACK packets. However, due to the use of $\mu$TESLA, there has to be either receiver side or sender side delay [48, 49]. The current version of Seluge adopts the cluster key approach due to its simplicity. We will explore the $\mu$TESLA approach in our future work.

### 3.2.3 Mitigating DoS Attacks against Signature Packets

All the previous secure code dissemination schemes [17, 14, 31] as well as Seluge use a signature to bootstrap the authentication of a code image. This signature is vulnerable to DoS attacks:
The adversary can inject bogus signature packets into the network, force the nodes that receive such packets to perform expensive signature verifications, and eventually exhaust their limited battery power.

Seluge adapts a recently developed weak authentication mechanism called *message specific puzzles* [47] to mitigate such DoS attacks [47]. This method has a setup phase before the deployment of sensor networks. During setup, the base station generates a one-way key chain consisting of $K_0, K_1, ..., K_n$, where $K_i = H(K_{i+1})$ ($i = n - 1, n - 2, ..., 0$) and $H(\cdot)$ is a cryptographic hash function. This is done by randomly selecting $K_n$ and repeatedly performing hash function $H$ to $K_n$, as shown in Figure 3.3. The base station then pre-distributes the *key chain commitment* $K_0$ to all sensor nodes before deployment. The keys $K_1, K_2, ..., K_n$ are called *puzzle keys*, and the puzzle key $K_i$ is used for the $i$th version of the disseminated code image.

![Figure 3.3: One-way key chain for puzzle keys.](image)

We use message specific puzzles to provide another layer of protection for the signature packet of each code image. For each version $i$, we use the puzzle key $K_i$ to generate a puzzle. Consider the signature packet of version $i$ code image, denoted as $i||M_i||\text{Sig}(i||M_i)$, where $i$ is the version number, $M_i$ represents the collection of the other fields in the signature packet, and $\text{Sig}(i||M_i)$ is the signature generated by the base station. The signature packet $i||M_i||\text{Sig}(i||M_i)$ and the puzzle key $K_i$ constitute a message specific puzzle. A valid solution $P_i$ is such a value that after applying the hash function $H$ to $i||M_i||\text{Sig}(i||M_i)||K_i||P_i$, the first $l$ bits of the resulting image are all “0”, as illustrated in Figure 3.4. The parameter $l$ determines the strength of the puzzle. Before transmitting the signature packet, the base station first tries to solve the puzzle by finding the puzzle solution $P_i$. Then the base station broadcasts the final signature packet $i||M_i||\text{Sig}(i||M_i)||K_i||P_i$.

Upon receiving a signature packet, each node first verifies that the puzzle key is valid using $H$ and $K_0$ (or a previously verified puzzle key) and that the puzzle key has not been used along with a valid signature before. Only when this verification is successful does the node verify the puzzle solution. If the puzzle solution is invalid, the receiver will simply drop the signature packet. Thus, without first solving some message specific puzzles with a fresh puzzle key, the adversary cannot force a node to verify signatures in forged packets.

Message specific puzzles can effectively mitigate DoS attacks against signature packets in code dissemination. The puzzle solution in each signature packet can be efficiently verified by a regular sensor node through a few hash function operations and comparisons. However, a puzzle
solution can only be found through brute-force search due to the one-way property of the hash function. Moreover, though it takes the same effort for both the base station and the adversary to solve a puzzle formed by a signature packet, the base station has a clear advantage over the adversary due to the prior knowledge of the puzzle keys: The base station has enough time to solve a puzzle off-line before disseminating a new code image. In contrast, the adversary has to solve puzzles after seeing a puzzle key but before the puzzle key becomes invalid when the signature packet reaches the target sensor nodes. Thus, with an appropriate puzzle strength, the message specific puzzle mechanism substantially increases the difficulty of launching DoS attacks against signature packets. Detailed analysis of message specific puzzles can be found in [47].

3.3 Analysis

3.3.1 Security Analysis

**Integrity of Code Images:** In Seluge, the trusted base station uses a digital signature to authenticate the root of the Merkle hash tree in page 0, with the private key only known to itself. All the sensor nodes know the public key of the base station, and thus can verify the signature. Under the assumption that the adversary cannot compromise the base station, it is guaranteed that all sensor nodes can authenticate any received signature packet as well as the root of the Merkle hash tree contained there. This means that all the nodes can authenticate the hash packets in page 0 once they receive such packets, based on the security of Merkle hash tree [42]. The hash packets include the hash images of the data packets in page 1. Thus, after verifying the hash packets, a sensor node can easily verify the data packets in page 1 based on the one-way property of cryptographic hash functions. Likewise, once verifying the data packets in page $i$, a sensor node can easily authenticate the data packets in page $i + 1$. 

![Figure 3.4: Message specific puzzles.](image-url)
where $i = 1, 2, \ldots, P - 1$. In summary, if the adversary injects a forged or modified code image, each receiving node can detect it easily because of the (immediate) authentication of code dissemination packets.

**Resistance to DoS Attacks:** As discussed in Section 1.1.1, there are three types of DoS attacks against Deluge based code dissemination: (1) DoS attacks exploiting authentication delays, (2) DoS attacks exploiting the expensive signature verifications, and (3) DoS attacks exploiting the Deluge propagation and suppression mechanisms.

Seluge is resistant to all three types of DoS attacks from external attackers. Due to the page-by-page dissemination strategy, upon receiving a packet, each node should have already received its hash image in the corresponding packet of the previous page (or in a hash packet in page 0). Thus, it can immediately authenticate any packet it receives in the current page, and successfully defeat DoS attacks exploiting authentication delays. Moreover, because of the use of message specific puzzles [47], each node can perform a few efficient hash function operations and comparisons to detect fake signature packets. Thus, Seluge provides resistance to DoS attacks that send fake signature packets. Finally, Seluge uses cluster keys to authenticate every advertisement or SNACK packet. As a result, an external attacker cannot convince regular nodes to misuse the propagation or suppression mechanisms.

Seluge can successfully defeat the first two types of DoS attacks even if there are compromised nodes. Indeed, without the private key and the unreleased puzzle keys on the base station, even an inside attacker cannot forge any code dissemination packets. However, Seluge cannot entirely prevent compromised nodes from launching the third type of DoS attacks that exploit the Deluge propagation and suppression mechanisms. A compromised node may misuse Deluge propagation and suppression mechanisms to mislead its neighbors. Fortunately, such DoS attacks are hard to coordinate and easy to detect, and the impacts are local to the compromised nodes. We will investigate how to detect such misbehaving nodes in the future.

### 3.3.2 Performance Analysis

**Communication Overhead:** We denote the code image as $CI$, and the maximum payload size per packet as $|payload|$. We first analyze the communication overhead to set up cluster keys between neighbor nodes, and then derive the communication overhead to disseminate a code image. We omit the analysis of overhead due to advertisement and request packets, because it remains the same as in Deluge.

As described in Section 3.2.2, each new node periodically broadcasts hello packets for a while to notify its neighbors. When a node hears a hello packet from a new neighbor, it sends its encrypted cluster key to the sender, and requests the sender’s encrypted cluster key back. This cluster key exchange phase is performed for a limited period of time for each node. Thus,
the total number of hello and cluster key packets that each node transmits is limited, though it varies depending on the actual deployment parameters.

The communication cost for propagating a code image includes a signature packet, hash packets, and data packets.

The total number of data packets depends on \( P \), the number of the pages. Given the parameters \( N \) (number of packets per page), \(|\text{payload}|\) (payload size available for code), and \(|H(\cdot)|\) (size of a hash image), \( P \) can be determined as follows: Since each packet in all the pages except for the last one should deliver a single hash value, each packet in page 1 to page \( P - 1 \) has \(|\text{payload}| - |H(\cdot)|\) bytes available for code, and each packet in page \( P \) has \(|\text{payload}|\) bytes all available for code. Thus, we can calculate \( P - 1 = \left\lceil \frac{|CI| - N \cdot |\text{payload}|}{|\text{payload}|} \right\rceil \). The number of packets in the last page can be calculated as \( \left\lceil \frac{|CI| - N \cdot (P - 1) \cdot |\text{payload}| - |H(\cdot)|}{|\text{payload}|} \right\rceil \), which may be less than \( N \). Thus, the total number of data packets is \( (N \cdot (P - 1) + \left\lceil \frac{|CI| - N \cdot (P - 1) \cdot |\text{payload}| - |H(\cdot)|}{|\text{payload}|} \right\rceil) \), where \( P - 1 = \left\lfloor \frac{|CI| - N \cdot |\text{payload}|}{|\text{payload}|} \right\rfloor \).

Now consider the number of hash packets in page 0, which is the number of leaves in the Merkle hash tree. As discussed in Section 3.2.1, the number of hash packets is \( M = 2^k \), where \( k \) is the minimum value that satisfies the following inequality: \( \frac{N \cdot |H(\cdot)|}{2^k} + k \cdot |H(\cdot)| \leq |\text{payload}| \).

Storage Overhead: Now we analyze the maximum buffer size required on each node. In Seluge, each node needs to authenticate advertisement and request packets from its neighbors using the right cluster keys. Thus, each node should store these cluster keys. Suppose each node keeps at most \( m \) cluster keys for its neighbors. Moreover, each node needs to store the hash images of the packets in the page to be received; such hash images are distributed in the corresponding packets in the previous page. Note that once a packet is received correctly, its hash image can be discarded, and the buffer entry can be reused for the hash image of the packet in the next page. Thus, each sensor node needs to have buffer for at most \( N \) hash images. In total, the maximum buffer size required by Seluge on each sensor node is \( m \times |K_c| + N \times |H(\cdot)| \), where \(|K_c|\) denotes the size of a cluster key.

Computation Overhead: Now we analyze the computation cost that Seluge requires on regular sensor nodes. Let us first consider attack-free cases. For each cluster key message, a sender adds message integrity code (MIC) for authentication and then encrypts the cluster key. A received cluster key packet is decrypted first and then verified with the MIC in the packet. Therefore, one MIC generation (or verification) and one encryption (or decryption) per transmission of a cluster key message are required on the sender (or receiver). Each advertisement or SNACK packet requires a MIC for authentication, and thus a sender (or a receiver) needs to generate (or verify) a MIC.

Consider the computation required to authenticate one code image. For each signature
packet, two hash operations are needed to verify the puzzle key and the puzzle solution, respectively, and a signature verification operation is performed. Each hash packet is verified by $\log M + 1 = k + 1$ hash operations, and the hash packets together require $M(\log M + 1)$ hash operations. Each of the remaining data packets is verified by a single hash operation. We already analyzed the total number of data packets earlier in this subsection. Thus, in attack-free situations, the total computation cost required to verify a single code image includes one signature verification and $(2 + M(\log M + 1) + N(P - 1) + \lceil |CI| - N(P - 1)|\text{payload}| - |H(\cdot)| \rceil)$ hash operations, where $P - 1 = \lceil \frac{|CI| - N|\text{payload}|}{N(\text{payload}) - |H(\cdot)|} \rceil$.

When there are attacks, a node being attacked must perform more computation. The actual computation depends on the volume of the attacks. However, as discussed in Section 3.3.1, the extra computations are mostly those that can be efficiently performed, such as hash operations.

### 3.3.3 Comparison with Previous Approaches

**Comparison with Sluice [31] and Berkeley approach [17]:** Sluice and the Berkeley approach have similar constructions as well as similar properties. They can prevent malicious code images from being accepted at sensor nodes. However, both of them are vulnerable to several types of DoS attacks. First, they are both vulnerable to DoS attacks exploiting authentication delays. As discussed in Section 1.1.1, the adversary can send a large number of bogus packets to exhaust the buffers at receiving nodes. Second, both of them overlooked the authentication of advertisement and SNACK packets. The adversary can attack sensor nodes by misusing the Deluge propagation and suppression mechanisms. Finally, there is no protection for the signature packet in either approach. This allows the adversary to exhaust the battery power on sensor nodes by sending a large number of forged signature packets. In contrast, as discussed in Section 3.3.1, Seluge can guarantee the code image integrity and deal with all these attacks.

**Comparison with Colorado approach [14]:** The Colorado approach can provide code image integrity protection. In addition, it allows each code packet to be immediately authenticated upon receipt, and thus is not vulnerable to DoS attacks exploiting authentication delays. Though the Colorado approach achieves the same property as Seluge, it is much less efficient than Seluge. The Colorado approach uses a per-page Merkle hash tree; a node transmits a request packet for each level in the tree, and waits for the packets only at the requested level. This essentially disrupts the efficient page-by-page propagation mechanism used by Deluge. As a result, this approach adds not only additional packets to transmit, but also additional propagation delay. In contrast, Seluge seamlessly integrates the Deluge page-by-page propagation mechanism.

As discussed in Section 1.1.1, the Colorado approach is vulnerable to online DoS attacks.
against the signature packets. Moreover, though the Colorado approach discussed the possibility of authenticating SNACK packets to partially address the DoS attacks exploiting the Deluge propagation and suppression mechanisms, it overlooked the authentication requirements for advertisement packets. Therefore, it is still vulnerable to similar DoS attacks. In contrast, Seluge can handle both types of DoS attacks gracefully.

3.4 Implementation and Experiments

3.4.1 Implementation

We implement Seluge as an extension to Deluge 2.0 in the current TinyOS distribution. Our implementation has both base station side and sensor side programs. The base station side programs are Java programs expected to run on a PC. They extend the Deluge Java tools to construct and inject new code dissemination packets into sensor networks. The sensor side program is written in nesC [19] and runs on regular sensor nodes.

We use the 64-bit truncation of SHA-1 as the hash function $H$. It provides sufficient pre-image resistance, and has been used previously (e.g., [17]). For digital signatures, we use ECDSA over the 160-bit elliptic curve secp160k1, which is defined in [8]. On the base station side, we use the JCE provider in the Bouncy Castle Crypto APIs [1] for hash function, key generation, and signature generation operations. On each sensor node, we integrate TinyECC [34] into Seluge to perform hash function and signature verification operations. Moreover, we use the hardware cryptographic support in the CC2420 radio component on MicaZ motes [3] for symmetric cryptographic operations, including the encryption (using AES) and authentication (using CBC-MAC) of cluster keys, and the authentication of page advertisement and SNACK packets.

We add the following functionalities in the Java tools on the base station side: Computation of the hash images of the data packets from the last to the first page, construction of the page 0 Merkle hash tree and then the hash packets from the hash images of the page 1 packets, and generation of the signature packet from the root of the Merkle hash tree and the meta data of the code image (e.g., version number, size). We include the message specific puzzle mechanism developed in [47] in both the Java tools and the sensor programs.

We add a PacketVerifier module into the Deluge nesC library to perform verification of signature packets (including both puzzle and signature verification), hash packets, and data packets. The commitment of the puzzle key chain used in message specific puzzles and the public key of the base station, which are generated by the Java tools, are pre-distributed to all nodes. The pairwise keys used to distribute cluster keys are also pre-distributed to all nodes.

Table 3.1 shows the ROM and RAM usage of Seluge on MicaZ motes. The code size of
Table 3.1: Code size (bytes) on MicaZ.

<table>
<thead>
<tr>
<th>Code</th>
<th>ROM</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deluge</td>
<td>22,226</td>
<td>1,123</td>
</tr>
<tr>
<td>Seluge</td>
<td>45,258</td>
<td>2,278</td>
</tr>
<tr>
<td>TinyECC in Seluge</td>
<td>13,044</td>
<td>426</td>
</tr>
</tbody>
</table>

Deluge and that of TinyECC are also included for reference purposes. It is easy to see that Seluge increases both the ROM and RAM consumption compared with Deluge, and the majority of the ROM increase is due to TinyECC.

3.4.2 Experimental Evaluation

We have provided theoretical analysis of the security and performance properties of Seluge in Section 3.3. In this subsection, we report the experimental evaluation of Seluge in a network of MicaZ motes [3]. For comparison purposes, we performed the same set of experiments with Deluge [24]. Moreover, as discussed in Section 1.1.1, the Berkeley approach [17] can be revised to mitigate the DoS attacks against dissemination packets. We obtained the source code from the authors of [17], made the revision, and used it in our experiments. Finally, we implemented the Colorado approach [14] and included it in our experimental evaluation. However, we did not include Sluice [31], since it offers much weaker security properties than the other approaches.

We use two performance metrics in our evaluation: Propagation delay and communication overhead. The propagation delay is the time required to finish disseminating a code image to all the nodes in the network. As mentioned in [24], for performance reasons, Deluge requires that every node keep its radio on. Thus, the propagation delay is closely related to the energy consumption required by code dissemination. The communication overhead is measured as the total number of packets transmitted by all the nodes during a code dissemination, which is also related to radio power consumption. Moreover, we also examine the propagation dynamics on individual nodes during the code dissemination to understand how each node receives different pages of the code image.

We perform the experiments in a testbed of 65 MicaZ motes. Figure 3.5 shows the layout of the testbed. The sensor nodes are deployed in 25 rooms, including offices, labs, server rooms, and corridors, covering an area of 152.5 \times 97 square feet. We equip each node with an Ethernet programming board, which provides remote access to the node. We only use the programming boards to gather evaluation results from the nodes; they do not interfere with the radio communication between sensor nodes at all. We set the transmission power level of
the radio module (CC2420) as $-3$dBm to increase the hop distance of the network.

Similar to Deluge, we need to configure a number of parameters for Seluge before code dissemination. We divide each code page into 48 packets, as the default setting in Deluge. To integrate the security mechanisms and the Deluge propagation mechanisms, we have to make certain changes to some Deluge parameters. Deluge uses a 2ms gap between two data packet transmissions. However, a SHA-1 hash verification operation takes about 15ms. Thus, we increase the transmission gap from 2ms to 17ms to accommodate this time requirement. Moreover, we increase the SNACK packet delay from 256ms to 1 second, so that a requesting node gives the sender enough time to transmit all the requested packets. Due to the dependency between the SNACK delay and advertisement delay, we also change the lower bound of the advertisement period to 2 seconds. The upper bound of the advertisement period remains the same default value of 60 seconds as in Deluge.

In these experiments, we use two different packet payload sizes, 102 bytes and 62 bytes, to examine the performance in different situations. (Note that the maximum payload size in IEEE 802.15.4 [27] is 102 bytes.) To investigate and compare the impact of disseminated code size on performance, we use four different code image sizes: 10K bytes, 20K bytes, 30K bytes, and 40K bytes. In each experiment, we inject a new code image at the star-shaped node located at the bottom-right corner in Figure 3.5. For each test case, we perform the same experiment 20 times and take an average over them.

**Propagation Delay**

Figure 3.6 shows the propagation delays of these schemes in the experiments. As the code image size increases, the propagation delays of all schemes increase almost linearly. Since the number
of packets required for a given code image increases as the packet payload size decreases, for all approaches, the propagation delays for 62 bytes payload size are longer than those for 102 bytes payload size.

Let us first compare the propagation delays in Seluge and the Colorado approach. For all the code image sizes, the propagation delays in Seluge are much less than those in the Colorado approach, and the gap between them becomes larger as the code image size increases. Among all the experiments, the average propagation delay of the Colorado approach is 63% longer than that of Seluge. When the packet payload size is 62 bytes, it takes the Colorado approach 30% to 82% longer time than Seluge to disseminate a code image. In the worst case, when the code image size is 20K bytes, the delay of the Colorado approach is about 82% longer than that of Seluge. Similarly, when the packet payload size is 102 bytes, it takes the Colorado approach 51% to 73% more time to finish disseminating a code image, where the worst case (i.e., 73%) happens when the code image size is 30K bytes.

![Figure 3.6: Propagation delay. (The Berkeley approach does not protect maintenance packets. When this mechanism was disabled in Seluge, the propagation delay was reduced by 30 – 146 seconds.)](image)

As we explained earlier, the main reason for this performance difference is that the Colorado approach propagates each code page and the corresponding per-page Merkle hash tree in a level-by-level fashion. This approach increases the interaction between a sending node and its receivers, and disrupts the page-by-page propagation in Deluge. In contrast, Seluge integrates the authentication and DoS-resistance mechanisms seamlessly with the Deluge page-by-page propagation and suppression mechanisms.
Let us now compare Seluge with Deluge. In all the experiments, Seluge introduces on average 21% longer propagation time than Deluge. When the packet payload size is 62 bytes, it takes Seluge 9% to 82% longer time than Deluge. In the case, when the code image size is 40K bytes, the delay of Seluge is even about 7% shorter than that of Deluge. When the packet payload size is 102 bytes, the propagation delay of Seluge is 1% to 29% longer than Deluge. The worst case scenarios in both packet payload sizes happen when the code image size is 10K bytes.

The additional delay introduced by Seluge is due to the propagation and verification of the signature packet, the dissemination of the (additional) hash packets, and the increase of the number of the data packets due to the inclusion of hash images. Nevertheless, as shown in Figure 3.6, the additional propagation delay introduced by Seluge is much smaller than that by the Colorado approach.

The (revised) Berkeley approach introduces slightly longer delay than Seluge. However, the Berkeley approach does not provide authentication of Deluge maintenance packets. To get better understanding of the performance difference between Seluge and the Berkeley approach, we run another set of experiments using Seluge with cluster key based local authentication disabled. Our results indicate that without local authentication of maintenance packets, the propagation delay in Seluge can be reduced by 56, 106, 112, and 146 seconds for code sizes 10K, 20K, 30K, and 40K when the packet size is 62 bytes, and by 30, 55, 77, 132 seconds for code sizes 10K, 20K, 30K, and 40K when the packet size is 102 bytes.

These experimental results demonstrate that Seluge introduces much less propagation delay into code dissemination than the Colorado approach, in addition to the stronger security properties.

**Communication Overhead**

Figure 3.7 shows the communication overheads of all these schemes, which are measured as the total number of packets transmitted by all the nodes in each test case. For the communication overheads of Seluge and the Colorado approach, we consider SNACK packets, hash packets (called index packets in the Colorado approach [14]), and data packets because those three types of packets are additionally required for a dissemination. Likewise, we consider SNACK and data packets for the communication overhead of Deluge and the Berkeley approach. As in the evaluation results for propagation delays, for all approaches, the communication overheads increase approximately linearly as the code image size grows, and the communication overheads for 62 bytes payload size are larger than those for 102 bytes payload size. In all the experiments, the Berkeley approach has the largest communication overhead, the Colorado approach is ranked the second, and Seluge has slightly larger communication overhead than Deluge. In particular,
the Berkeley approach has more than twice as much overhead than any other approach.

![Graph showing communication overhead](image)

**Figure 3.7: Communication overhead.**

**Propagation on Individual Nodes**

We also investigate how code pages are propagated on individual nodes to get more insights. In the following, we select two nodes in the testbed, which are marked as $n_1$ and $n_2$ in Figure 3.5, to see how they receive code pages over time. (We select these two nodes to present, because $n_1$ is close to the source and $n_2$ is far away from the source. They are expected to have different situations during code dissemination.) In the following, we show the dynamic propagation features on these two nodes, using the test case where we inject a code image of 30K bytes with 102 bytes payload size.

Figure 3.8 shows the time points when $n_1$ or $n_2$ finishes receiving every page of the code image under all approaches. The x-axis represents the completion time for a page, and the y-axis represents the ratio of the number of completed pages to the total number of pages in a code image. As time goes on, $n_1$ and $n_2$ gradually complete the receiving of the code image. Due to the effect of spatial multiplexing, $n_2$ receives some pages of the code image before $n_1$ finishes to receive all the pages of the code image.

Figure 3.8 confirms at individual node level that Seluge allows much faster propagation than the Colorado approach and the Berkeley approach. Compared to Deluge, Seluge shows very similar dissemination dynamics on both of the nodes. Moreover, on these two nodes, Seluge even surpasses Deluge from certain time points and finally completes the code image earlier than Deluge, though it involves additional security mechanisms such as signature verification.
and hash packet distribution.

### 3.5 Summary

In this chapter, we presented the design, implementation, and evaluation of Seluge. Besides the efficiency and robustness inherited from Deluge, Seluge provides security protections for code dissemination, including the integrity protection of code images and resistance to the DoS attacks that exploit the code dissemination protocol. Seluge is superior to all previous attempts for secure code dissemination, and is the only one that seamlessly integrates the security mechanisms and the Deluge efficient propagation strategies.
Chapter 4

Mitigating Wireless Jamming Attacks via Channel Migration

In this chapter, we investigate a new approach called channel migration to mitigating wireless jamming attacks. For resilience against jamming attacks, our scheme exploits the multiple wireless channels typically available on most wireless platforms. Each node estimates the qualities of the channels that it has a chance to observe. If it detects poor quality on its main communication channel, it leaves for a different channel for communication. As a result, a node currently on a jammed channel can continue communication with its neighbors on other available channels. A nice property of the channel migration scheme is that it does not depend on any single, fixed channel and executes in a decentralized and independent way on each wireless node.

To improve channel synchronization between neighbors, we develop an advertisement mechanism, which enables each node to periodically inform its neighbors using different communication channels about its current main communication channel. Such a mechanism enables each node to keep track of the statistics of different channels. When switching its main communication channel, each node uses the channel statistics to select the one where it possibly gets the most benefit.

To investigate the effectiveness of the proposed channel migration scheme, we perform a case study to apply the channel migration scheme to Seluge [26], a secure code dissemination system for wireless sensor networks. We name the resulting protocol Jamming-Resilient-Seluge, or simply JR-Seluge. We perform a theoretical analysis of the anti-jamming capability of JR-Seluge and also run experimental evaluation in a testbed of wireless sensor network.

The rest of this chapter is organized as follows. Section 4.1 clarifies our assumptions and jamming model. Section 4.2 presents the design of our channel migration scheme and gives a qualitative analysis of its anti-jamming properties. Section 4.3 presents the case study with
JR-Seluge and provides a theoretical analysis of JR-Seluge’s jamming resilience property and the experimental evaluation of JR-Seluge. Section 4.4 summarizes this chapter.

4.1 Assumptions and Jamming Model

We assume that each wireless node has access to multiple channels and can switch channels at run time. This is true on most wireless devices, including wireless sensor platforms (e.g., Mica2 [12], MicaZ [3], TelosB [4]). For simplicity, we assume that each node is equipped with a single radio; that is, it can transmit and/or receive on only one channel at a time. But our approach can be easily extended to the case that each node is equipped with multiple radios. We also assume that each node is running an upper layer protocol that is responsible for communicating with the neighbors using the same communication channel as itself.

We assume that every node is able to authenticate packets broadcast by its neighbor nodes, for example, using either cluster key-based approach [26] or µTESLA-based approach [61]. For the use of the cluster key-based approach, we further assume that wireless nodes are able to establish pairwise keys between neighbor nodes, for example, using one of the existing schemes (e.g., [9, 37, 15, 70, 38]). For the use of the µTESLA-based approach, we further assume that wireless nodes are able to securely synchronize their clocks, for example, using one of the existing schemes (e.g., [18, 41, 56, 61]). To prevent replay attacks, each packet includes a unique sequence number. We also assume that a sender and receiver can encrypt and decrypt packets, respectively, using the same key used for local broadcast authentication.

The goal of a jammer is to prevent the wireless nodes within her signal range from receiving messages from their neighbor nodes. We assume that each jammer has the ability to sense and jam multiple channels of her choice concurrently, but cannot jam all available channels at the same time. In addition, we assume that the jammer dynamically switches channels she senses and jams, but stays on a channel at least for a certain amount of time, during which legitimate nodes on other available channels can transmit and receive one or more packets.

A jammer may constantly, randomly, or reactively jam multiple channels at the same time [68]. In a constant jamming, the jammer continuously transmits jamming signals on select channels. In a random jamming, the jammer emits jamming signals at random times (instead of continuously emitting jamming signals) on randomly chosen channel(s). In a reactive jamming (a.k.a. responsive jamming), the jammer senses transmission on one or several channels, and starts jamming a channel for a period of time when she senses actual transmission on it. To maximize the number of affected channels, the jammer may sense and/or jam multiple channels at the same time. The jammer may capture a wireless node in the network to get information useful for jamming (e.g., secret hopping patterns).
4.2 Mitigating Wireless Jamming Attacks via Channel Migration

In this section, we propose a channel migration technique to allow wireless nodes that have access to multiple wireless channels to defend against jamming attacks. The basic intuition is to fall back to alternative channels when the main channel currently used for wireless communication is jammed. The key contribution of channel migration is a suite of protocol components that allow wireless nodes to figure out when and what channel to fall back to in order to avoid jamming.

For the sake of presentation, we first clarify a few terms. If two nodes are within each other’s communication range, we say that they are neighbor nodes, or simply neighbors. The primary channel for a wireless node is a channel where the node stays for transmitting and/or receiving data packets with its neighbors. In the case that the node is equipped with multiple radios, it has multiple primary channels. Each node stays on its primary channel for at least a channel switching period. All the remaining channels except for the primary channel are called secondary channels. A node has an active task if it is transmitting/receiving, or scheduled to transmit/receive data packets. A node can have an active task only on the primary channel, and a node having an active task does not leave the primary channel before finishing the active task as long as it does not experience poor channel quality on the primary channel. If a node can possibly have an active task by switching its primary channel to a secondary channel, we say that there is a potential active task for the node on that secondary channel.

4.2.1 Overview of Channel Migration

Let us discuss the overall behavior of channel migration. In addition to communicating data and/or control packets with its neighbors as required by upper layer protocols, each wireless node also broadcasts advertisement packets to notify its neighbors about its identity, its primary channel and the remaining channel switching period on that primary channel. In order to cope with potential jamming attacks, each node advertises on its primary channel as well as its secondary channels. Each node also switches its primary channel adaptively, particularly when its primary channel is jammed.

Each node estimates the quality of the channels on which it receives packets. If it detects that the quality of the primary channel is not good enough, it switches the primary channel to a different one. Moreover, the node keeps track of its latest quality assessments for all channels. When selecting a secondary channel to advertise or the new primary channel, the node considers these channel quality assessments.

A node always communicates with its neighbors on its primary channel. Moreover, each
node pays attention to advertisements from its neighbor nodes with different primary channels. When it overhears such an advertisement packet, it first checks if it has experienced poor quality lately on that secondary channel (specified in the primary channel field of the advertisement). Only if the answer is “no” does the node further check whether it has a need to communicate with the neighbor on that secondary channel. If there is a potential active task, the node switches its primary channel to the secondary channel only if it has no neighbor with whom it needs to communicate on its primary channel.

If the node has no active task for a channel switching period, it switches its primary channel. When a node finishes an active task (either sending or receiving), it resets its channel switching period. In other words, the node stays on its current primary channel longer, since there may still be work to do.

Intuitively, when the primary channel of a node is under jamming, the node will experience poor quality on its primary channel or have no active task for a channel switching period and thus leave from the primary channel. If there is no jamming on the new primary channel, the node will overhear advertisements from nodes either on the primary channel or other secondary channels. In either case, the node will find new active tasks on the primary or a secondary channel, and data communication can continue there.

In the following, we describe channel migration in detail.

4.2.2 Channel Quality Estimation

Estimation of channel quality is essential to the proposed channel migration scheme. The decision for a node to switch its primary channel, the choice for the next primary channel, and the selection of a secondary channel to advertise are all based on the estimated qualities of the channels that the node has a chance to observe. A node estimates the quality of the channel it uses by estimating how many of the packets that its neighbor nodes have transmitted on the channel have been successfully received.

There are multiple possible ways to perform channel quality estimation. We adopt a simple method proposed in [68]. Whenever a node overhears a packet (or preamble), the node checks whether the packet passes the CRC check. By observing the ratio of the number of packets that pass the CRC check to the total number of packets overheard, the node can estimate the channel quality. For the sake of presentation, we refer to this ratio as Valid Packet Ratio (VPR).

Each node maintains a list of VPR values for all channels. We use a threshold $T_{VPR}$ for channel quality. If the VPR measurement of a channel is less than $T_{VPR}$, the node concludes that the quality of the channel is poor. Initially, the VPR values of every channel are set into $T_{VPR}$. In order to acquire a stable and reliable estimate, we require that the node can get a
VPR measurement on a channel only after it has overheard at least $n$ packets on the channel within a certain period of time. Moreover, if the node acquires no new VPR measurement on a channel for a certain period of time, it sets the VPR value of the channel to $T_{VPR}$. The node refers to the VPR values when it selects a channel to advertise or migrate.

4.2.3 Advertisement

Each node periodically broadcasts advertisement packets on the primary or secondary channels. Each advertisement packet contains the node’s identity, the primary channel number, and the remaining channel switching period on that primary channel. Intuitively, such advertising allows the node to notify its neighbor nodes with different primary channels on which channel it currently resides and how long it will stay there, and by monitoring advertisements from neighbors, each node can keep track of how many neighbors are on each channel and use the statistics for its future channel decision.

Advertisement in Primary and Secondary Channels

When a node is initialized, it starts two separate time periods, called the primary channel advertisement period and the secondary channel advertisement period, respectively. Each primary and secondary channel advertisement period consists of two halves. At the beginning of each primary and secondary channel advertisement period, the node selects a random time point in the first half of the period, when it broadcasts the advertisement packet of the current period. For all the remaining advertisement period, the node monitors its neighbors’ advertisements, staying on its primary channel. Such randomized advertising is for increasing the chance for neighboring nodes to receive each others’ advertisements.

At the selected random time point in each primary channel advertisement period, the node broadcasts an advertisement on the primary channel. But if the node is performing an active task now, it suppresses the current primary channel advertisement packet. At the selected random time point in each secondary channel advertisement period, the node first decides on which secondary channel it will advertise. If the node cannot find any channel to advertise on, or if it is performing an active task now, it suppresses the current secondary channel advertisement packet. Otherwise, it temporarily switches its radio to the selected secondary channel, transmits an advertisement packet, and switches the radio back to the primary channel. After finishing advertising on a channel, the node marks the channel as advertised and does not advertise in that channel again for next $l$ secondary channel advertisement periods. In this way, the channel migration scheme prevents the node from advertising in the same channel too frequently.
Selecting a Secondary Channel for Advertisement

When selecting a secondary channel to advertise, if a node has a specific need to communicate with one particular node and it already knows the channel where the intended node currently resides, the node will select the channel. If this is not the case, in general, the node refers to the statistics of the number of neighbors on each channel and gives higher priority to those where there are more neighbors with whom it may communicate. This is to maximize the possibility that more nodes will benefit from this advertisement (by, e.g., switching to this channel). In addition, secondary channels with better quality have higher priority to be selected, because the better the quality of a channel is, the higher chance an advertisement on the channel is successfully delivered.

In order to select a secondary channel to advertise, each node performs the following procedure, excluding the secondary channels on which it advertised during the most recent \( l \) secondary channel advertisement periods. The node first looks for the secondary channels where there are the most neighbors, and then the one with the largest VPR among them. But if the node does not find any channel with one or more neighbors, then it considers the status of the current primary channel. If the upper layer protocol indicates that on the current primary channel there is at least one neighbor with whom the node needs to communicate, the node concentrates on the need for communication on the primary channel without temporarily leaving the primary channel for secondary channel advertisement. Otherwise, if there is no need for communication on the current primary channel, the node selects the secondary channel with the largest VPR. Note that whenever the node has multiple candidates meeting the condition in the end, it randomly chooses one of them.

Intuitively, through advertising on the selected channel, a node attempts to attract potential receivers or senders on its secondary channels to its primary channel.

4.2.4 Switching the Primary Channel

The ability to switch the primary channel is critical in mitigating jamming attacks. It allows nodes to evade the jammed channel and continue communication with their neighbors on other available channels. In the following, we first explain how the proposed channel migration scheme decides when to switch a node’s primary channel and then describe how a node determines which channel to switch to.

Making Channel Switching Decision

Based on its observation of the current primary channel, node \( A \) considers switching its primary channel in the following three cases:
Case C1 represents the situation where node A realizes the existence of the potential active task(s) on Ch_B. Now consider case C2. As mentioned at the beginning of this section, after finishing an active task, a node resets its channel switching period to stay longer on the current primary channel, because there may be more active tasks to do. Thus, case C2 implies that node A has never had an active task at least for a channel switching period on its current primary channel. In such a case, node A may switch its primary channel. Finally, in case C3, node A realizes that the quality of its current primary channel is poor and thus tries to switch to a channel with better quality.

In case C1, if node A has recently experienced poor quality on Ch_B (node A’s VPR value of Ch_B < T_{VPR}), it just ignores the received advertisement. Otherwise, node A checks the current status of Ch_A through the following procedure to decide whether it leaves Ch_A. If node A is performing an active task now, or if the upper layer protocol indicates that node A has a need to communicate with at least one neighbor on Ch_A, it does not leave Ch_A since active tasks on the primary channel have higher priority than any potential active tasks on secondary channels. Otherwise, it leaves Ch_A right away. In case C2, node A also decides whether to leave Ch_A through the above procedure. In addition, if node A has experienced good quality on Ch_A lately, it does not leave Ch_A since it may still be able to communicate with its neighbors using Ch_A. In case C3, node A leaves Ch_A right away due to the poor channel quality.

Selecting the New Primary Channel

When selecting the new primary channel, each node gives higher priority to the secondary channels where it has not experienced poor quality lately and there are more neighbors with whom it may need to communicate.

Suppose node A has decided to leave Ch_A. In case C1, the new primary channel is simply Ch_B, B’s primary channel. In cases C2 and C3, node A first looks for the secondary channels whose VPR values are greater than or equal to T_{VPR}. If no such secondary channels exist, node A simply selects the secondary channel with the largest VPR as its new primary channel. Otherwise, among all such secondary channels (whose VPR values ≥ T_{VPR}), it searches for the secondary channel where there are the most number of neighbors with whom node A may need to communicate. If there are multiple candidates, it randomly chooses one of them. Intuitively,
this is for finding a good quality secondary channel where there are the most neighbors that node A can communicate with.

Figure 4.1: Illustration of our channel migration scheme

Figure 4.1 illustrates an example of the channel migration scheme. In this example, nodes 1 and 2 have needs to communicate with each other. They are initially on the same primary channel (Ch2) and so performing active tasks there. In the meantime, jamming occurs on Ch2 at time t1. Due to the jamming, nodes 1 and 2 experience poor quality on Ch2, and then migrate into Ch1 and Ch4 at times t2 and t3, respectively. After some time, at time t4 node 1 performs a secondary channel advertisement on Ch4, which is node 2’s current primary channel. Once node 2 overhears the advertisement from node 1, it switches its primary channel to Ch1 for the potential active task with node 1 there.

4.2.5 Analysis

Channel migration is a general approach for mitigating wireless jamming and its exact performance also depends on how it is integrated with the actual protocols. Thus, in this subsection, we give a qualitative analysis of its anti-jamming properties. In the next section, we will integrate channel migration into a code dissemination protocol, where we will examine its performance in a real application.

First of all, if a channel is jammed, the nodes using the channel as the primary channel will leave the channel after either detecting poor quality or staying there for at most a channel switching period. Next, when selecting the next primary channel, each jammed node independently makes the decision based on its own observations of the number of neighbors and the channel quality on every channel. It is hard for the adversary to precisely predict such local observations of the node and so the node’s channel decision. Moreover, each jammed node may select a different channel due to the difference on their local observations. Thus, it is non-trivial for the adversary to sustain its jamming on all those nodes by following their channel changes.
Finally, whenever a node co-locates on the same primary channel with a neighbor that it needs to communicate with, it can perform active tasks as long as the primary channel is not under jamming, without the need to wait for other neighbors to get aligned to its primary channel. Thus, communication may be concurrently performed on multiple channels. Consequently, the impact of jamming certain channels is temporary and limited to only those using the jammed channels as their primary channels.

### 4.2.6 Discussion

In spite of good anti-jamming properties, our channel migration scheme still has some limitations. First, if two neighboring nodes do not have any common available channel, they cannot communicate with each other in our scheme. Second, though authentication and encryption of advertisement packets are assumed, insider jammers, who capture keys for authentication and encryption from compromised nodes, can not only broadcast fake advertisements but also decrypt and observe advertisements from legitimate nodes whose keys are captured. Finally, if a jammer’s objective is to jam a specific node, she may analyze the packets transmitted by the target node and follow it to whatever channels the targeted node migrates to. As a result, the jammer can still jam the communication of the target node. Nevertheless, unless the jammer can jam all channels simultaneously, she cannot disable the wireless communication by all nodes. Other nodes can still use the channel not targeted by the jammer to communicate.

### 4.3 Case Study: JR-Seluge

To evaluate our channel migration scheme, we apply our scheme to Seluge [26], a secure code dissemination protocol for wireless sensor networks, in a case study. We name the resulting protocol Jamming-Resilient-Seluge, or simply JR-Seluge. Code dissemination is the process for propagating a new code update into the entire network through the wireless network formed by the sensor nodes in the network.

#### 4.3.1 Seluge Overview

Seluge adopts a page-by-page dissemination strategy from Deluge [24]. A code image is divided into fixed-size pages, and each page is further split into same-size packets. Each sensor node can request a page only after completely receiving every packet in the previous page. Seluge uses Trickle [33] for efficient advertisement of code metadata. Each node periodically advertises the version of its code image and the number of pages it has for that version. Once a node overhears an advertisement from a neighbor that has more pages than itself, it sends a request
to the neighbor. Upon receiving the request, the neighbor node broadcasts the requested data packets.

To protect integrity of a disseminated code image, Seluge authenticates every code dissemination packets. In addition, Seluge provides cluster key based local broadcast authentication for control packets such as advertisements and requests to prevent DoS attacks using fake control packets. Moreover, all kinds of packets in Seluge can be immediately authenticated upon receipt, which eliminates potential threats due to delayed authentication.

4.3.2 Integrating Channel Migration into Seluge

In addition to the number of neighbors currently residing on each channel, the code image versions and the number of pages those neighbors have are also important for selecting a channel in JR-Seluge. For this reason, we integrate our channel migration scheme into the advertisement scheme used by Seluge. Each advertisement packet in JR-Seluge contains two additional items: the latest code image version and the number of pages for that version the advertising node has. Each node keeps track of the latest version and the minimum and maximum page indices available on each channel along with the number of neighbors there, and refers to them to select a channel to advertise or migrate.

Advertisement Period

In Seluge, advertisement rate is dynamically adjusted for energy efficiency. If a node discovers its own advertisement is different from those received from its neighbors, it increases its advertisement rate. Otherwise, it decreases its advertisement rate. Using such dynamic adjustment of advertisement rate, Seluge can achieve rapid propagation during dissemination of a new code image, but consumes little resource in steady state.

We adopt the above idea to the advertisement mechanism in JR-Seluge to make it more efficient. Application to the primary channel advertisement is straightforward; If a node overhears an advertisement about a different version or a different number of pages from the neighbor using the same primary channel, it reduces the next primary channel advertisement period to the pre-defined minimum. Otherwise, the node doubles the next primary channel advertisement period.

Similarly, if the node overhears a different advertisement from a neighbor using a different primary channel (one of the node’s secondary channels), it sets the next secondary channel advertisement period to the pre-defined minimum. In this case, there may be neighbors with different versions or different number of pages on its secondary channels. By increasing its secondary channel advertising frequency, the node tries to notify its own existence to those neighbors as soon as possible.
We propose an additional heuristic to decide whether to increase the secondary channel advertisement period. If the node has not overheard any different advertisement packet from neighbors on its secondary channels during the current secondary channel advertisement period, the node checks the following two conditions to determine whether to increase the next secondary channel advertisement period. First, the node checks whether it has ever detected a neighbor that has something it needs to receive. Second, it checks if it currently has something to receive on its primary channel. Only if both conditions are satisfied does the node double its secondary channel advertisement period. In this way, the node knows that there is something to receive from one of its neighbors on one of its secondary channels. Without reducing its secondary channel advertising frequency, it does its best to find out on which channel it can receive data.

Channel Migration & Selection

We customize the procedure for each node to decide whether it leaves the current primary channel in each of cases $C_1$ to $C_3$ in Section 4.2.4. The primary objective of code dissemination is to update to the latest version. Thus in JR-Seluge if the overheard advertisement in case $C_1$ indicates that node $B$ has a newer version than the latest version available on $Ch_A$, node $A$ moves to $Ch_B$ right away regardless of the existence of an active task on $Ch_A$. Moreover, even if node $A$ currently has neighbors that need help from itself on $Ch_A$, it can leave $Ch_A$ for its own need with a certain probability ($0 \leq p_1 \leq 1$) if $Ch_A$ has at least one neighbor that can serve those neighbors instead of itself.

Next, we customize the procedures for selecting the new primary channel and a secondary channel to advertise. When switching the primary channel, each node tries to migrate into the channel where a potential sender with the most pages for the latest version it needs exists. For potential receivers with older versions or fewer pages than itself, it will try to attract them later to the new primary channel through secondary channel advertisements. But if they already have a potential sender on their primary channel, we exclude the corresponding channel, because they can receive the data from the potential sender there. Now we describe each procedure in detail.

When selecting the new primary channel, the node first looks for the secondary channel with a VPR value greater than $T_{VPR}$ where the latest version and the most pages it needs are available. Only when it finds no channel, it further looks for the secondary channel with a VPR value greater than $T_{VPR}$ where there are the most potential receivers. If it still finds no channel, it selects the secondary channel with the largest VPR.

To select a secondary channel to advertise, the node first checks the current status of its primary channel. If the node already has a potential sender on its primary channel, it only tries to attract potential receivers to its primary channel. Thus it selects the secondary channel with
the largest VPR among those where there are the most potential receivers. Otherwise, it needs to find potential senders as well as potential receivers. Thus it first looks for the secondary channel above mentioned, and only when it finds no channel, it further looks for the secondary channel with the largest VPR among those where the latest version and the most pages are available. If the node still finds no channel, and if it has no neighbors with whom it needs to communicate on its primary channel, it selects the secondary channel with the largest VPR. Note that if the node finds no channel in the end of this procedure, it suppresses the current advertisement packet.

4.3.3 Analysis

In this subsection, we provide a theoretical analysis of JR-Seluge’s anti-jamming capability. An experimental evaluation of JR-Seluge is also given in the next subsection.

We assume that each jammer is randomly located in the network. Each jammer follows the strategy described in the jamming model in Section 4.1. If a channel is jammed, it is impossible for regular sensor nodes within the jamming range to communicate with each other on that channel. All sensor nodes are randomly distributed over the entire network.

Let \( c \) represent the total number of channels in the system (\( c \geq 1 \)). Let node \( A \) denote any single node in the network. We assume that there are \( m \) (\( m \geq 1 \)) neighbors around node \( A \) that have a new code image \( A \) needs to download. There are also \( n \) (\( n \geq 1 \)) jammers whose jamming ranges cover node \( A \)’s and the \( m \) neighbors’ locations. We assume that jammer \( i \) (= 1, . . . , \( n \)) can jam at most \( f_i \) channels at the same time.

Let us divide the time line into a series of time units. For simplicity, we assume there is no channel switching on either sensor nodes or jammers within a single time unit. We also assume that if any two sensor nodes meet on the same channel available to both of them in a certain time unit, they can communicate \( k \) (\( k \geq 1 \)) code dissemination packets for the time unit. Let us assume that the new code image consists of \( l k \) packets (\( l \geq 1 \)).

**Lemma 4.1** The probability that node \( A \) meets at least one of the \( m \) neighbors on the same channel in a certain time unit is

\[
P_{\text{meet}} = 1 - \left( \frac{c-1}{c} \right)^m, \text{ where } c \geq 1 \text{ and } m \geq 1.
\]

**Proof:** The probability that node \( A \) meets none of the \( m \) neighbors on the same channel in a certain time unit is \( \left( \frac{c-1}{c} \right)^m \). Hence, the probability that node \( A \) meets at least one of them on the same channel in a time unit is \( P_{\text{meet}} = 1 - \left( \frac{c-1}{c} \right)^m \).

**Lemma 4.2** Given that node \( A \) meets at least one of the \( m \) neighbors on the same channel in a certain time unit, the probability that the channel is available to both of them for the time
unit is
\[ P(\text{avail}|\text{meet}) = P_{\text{avail}} = \{1 - \min(\frac{\sum_{i=1}^{n} f_i}{c}, 1)\}, \text{where } n \geq 1. \]

**Proof:** Since the event that node A meets at least one of the \( m \) neighbors on the same channel in a certain time unit is statistically independent of the event that the channel is available to both of them for the time unit, \( P(\text{avail}|\text{meet}) = P(\text{avail}) = P_{\text{avail}}. \) The \( n \) jammers can jam at most \( \sum_{i=1}^{n} f_i \) channels at the same time. Thus, the probability that the channel where node A meets the neighbor(s) is available to both of them for the time unit is \( P_{\text{avail}} = 1 - \min(\frac{\sum_{i=1}^{n} f_i}{c}, 1) \)

**Lemma 4.3** The probability that node A completely receives the new code image within \( N \) (\( N \geq 1 \)) time units is
\[ P(X \geq l) = \sum_{i=l}^{N} \binom{N}{i} P_k^i (1 - P_k)^{N-i}, \]
where \( P_k = P_{\text{meet}} P_{\text{avail}} \) and random variable \( X \sim \text{Bin}(N, P_k) \).

**Proof:** According to our assumption, if none of the \( n \) jammers does not jam node A’s primary channel in a certain time unit, and if at least one of the \( m \) neighbors co-locates on the same primary channel with node A in that time unit, node A can receive \( k \) packets for the time unit. Therefore, the probability that node A receives \( k \) packets for a certain time unit is \( P_k = P(\text{meet} \& \text{ avail}) = P_{\text{meet}} P_{\text{avail}} \) since the two events (meet and avail) are independent.

Let \( X \) be the random variable that represents the number of time units among \( N \), for each of which node A can receive \( k \) packets. Then \( X \) follows the binomial distribution with parameters \( N \) and \( P_k \) (\( X \sim \text{Bin}(N, P_k) \)). To receive the complete new code image within \( N \) time units, node A needs to be able to receive \( k \) packets in at least \( l \) time units of \( N \) time units. Hence, the probability that node A completely receives the new code image within \( N \) time units is \( P(X \geq l) = \sum_{i=l}^{N} \binom{N}{i} P_k^i (1 - P_k)^{N-i} \).

**Theorem 4.1** \( \lim_{N \to \infty} P(X \geq l) = 1 \) if and only if \( \sum_{i=1}^{n} f_i < c. \) That is, node A eventually receives the new code image if and only if \( \sum_{i=1}^{n} f_i < c. \)

**Proof:** Let us first prove that if \( \sum_{i=1}^{n} f_i < c, \lim_{N \to \infty} P(X \geq l) = 1. \lim_{N \to \infty} P(X \geq l) = 1 \) is equivalent to \( \lim_{N \to \infty} P(X \leq (l-1)) = 0 \) since \( \lim_{N \to \infty} P(X \geq l) = 1 - \lim_{N \to \infty} P(X \leq (l-1)). \) So, in the following, we prove that if \( \sum_{i=1}^{n} f_i < c, \lim_{N \to \infty} P(X \leq (l-1)) = 0. \)

Since the exponential function is monotonically increasing, \( P(X \leq (l-1)) = P(-X \geq -(l-1)) = P(e^{-X} \geq e^{-(l-1)}) \). By Markov’s inequality,
\[ 0 \leq P(e^{-X} \geq e^{-(l-1)}) \leq e^{l-1} E[e^{-X}]. \]
Since the moment-generating function of the binomial distribution \( X \) is 
\[ M_X(v) = E[e^{vX}] = \{P_k e^v + (1 - P_k)\}^N \ (v \in \mathbb{R}), \]

\[ 0 \leq P(X \leq (l - 1)) = P(e^{-X} \geq e^{-(l-1)}) \leq e^{l-1}\{P_k e^{-1} + (1 - P_k)\}^N. \]

If we take the limitation,

\[ 0 \leq \lim_{N \to \infty} P(X \leq (l - 1)) \leq e^{l-1} \lim_{N \to \infty} \{1 - P_k(1 - e^{-1})\}^N. \]

If \( \sum_{i=1}^{n} f_i < c \), \( 0 < \min(\frac{\sum_{i=1}^{n} f_i}{c}, 1) < 1 \) and so \( 0 < P_{\text{avail}} < 1 \). Since \( 0 < P_{\text{meet}} < 1 \), \( 0 < P_k = P_{\text{avail}} P_{\text{meet}} < 1 \). Then \( 0 < \{1 - P_k(1 - e^{-1})\} < 1 \). Hence, \( \lim_{N \to \infty} \{1 - P_k(1 - e^{-1})\}^N = 0 \). As a result, if \( \sum_{i=1}^{n} f_i < c \), then

\[ \lim_{N \to \infty} P(X \leq (l - 1)) = 0 \]

since \( 0 \leq \lim_{N \to \infty} P(X \leq (l - 1)) \leq 0 \).

Next, we prove that if \( \lim_{N \to \infty} P(X \geq l) = 1 \), \( \sum_{i=1}^{n} f_i < c \). This statement is equivalent that if \( \sum_{i=1}^{n} f_i \geq c \), \( \lim_{N \to \infty} P(X \geq l) \neq 1 \). If \( \sum_{i=1}^{n} f_i \geq c \), \( \min(\frac{\sum_{i=1}^{n} f_i}{c}, 1) = 1 \). Then \( P_k = 0 \), and so \( P(X \geq l) = 0 \). Thus \( \lim_{N \to \infty} P(X \geq l) = 0 \neq 1 \).

**Theorem 4.2** Assume that a given network is finite, there are one or more source nodes with a new code image, each sensor node has at least one communication path from a source node, and \( \sum_{i=1}^{n} f_i < c \) is guaranteed in any neighborhood over the entire network. Then every node in the network eventually receives the new code image.

**Proof:** Let \( D \ (D \geq 1) \) represent the maximum hop distance of the network from the source nodes. By theorem 4.1, every node in the first hop eventually receives the new code image from the source nodes. Let us assume that every node in the \( d \)th \( (1 < d < D) \) hop receives the new code image. Then it is also guaranteed by theorem 4.1 that every node in the \( (d + 1) \)th hop eventually receives the new code image from the nodes in the \( d \)th hop. As a result, every node in the entire network eventually receives the new code image.

### 4.3.4 Experimental Evaluation

In this subsection, we report the experimental evaluation of JR-Seluge in a testbed of wireless sensor network. We first compare the performance overhead of JR-Seluge with Seluge without and with jammers, and then further evaluate the jamming resilience of JR-Seluge in different jamming scenarios.
Experimental Setup

We perform the experiments in a sensor network testbed with 72 MicaZ motes. Figure 5.4 shows the layout of the testbed.

![Figure 4.2: The sensor network testbed (72 MicaZ motes; 152.5 feet × 97 feet).]

The testbed area includes offices, labs, server rooms, and corridors, covering an area of 152.5 feet × 97 feet. We equip each node with an Ethernet programming board, which provides remote access to the node. We only use the programming boards to gather evaluation results from the sensor nodes; they do not interfere with the radio communication between sensor nodes at all. We set the transmission power level of the MicaZ radio module (CC2420) as 0dBm.

We use several nodes in the testbed (the square-shaped nodes with numbers in Figure 5.4) as jammers. For the jammers, we modify the CC2420 Radio module in TinyOS so that a jammer keeps sending packets without clear channel assessment (CCA). With this jamming program, each jammer continuously injects noise into a specific channel throughout each experiment.

**Evaluation Metrics:** We use two metrics in our evaluation: *completion time* and *communication overhead*. The completion time is the time required to finish disseminating a code image to all the nodes in the network in each experiment. The communication overhead is measured as the total number of packets transmitted by all the nodes during a code dissemination. For communication overhead, we show the number of request and data packets and the number of advertisements separately.

**Evaluation Parameters:** Since in a real deployment different sensor nodes may stay on different channels at the time when code dissemination occurs, we configure JR-Seluge so that each node starts with a randomly selected channel as the initial primary channel. For Seluge,
we use channel 26, which does not overlap with 802.11 [11].

The settings for other parameters of JR-Seluge are as follows: The lower and the upper bounds of the primary and secondary channel advertisement periods are 512 ms and 2 minutes, respectively. We set the channel switching period to two primary channel advertisement periods. We set the least number of packets required for channel quality assessment as \( n = 100 \) and the channel quality threshold \( T_{VPR} = 0.65 \). That is, each node measures a VPR every 100 start symbols overheard, and if the VPR measurement is less than 0.65, the node leaves the current primary channel. We set \( p_1 = 0.7 \).

In each experiment, the dissemination starts from the star-shaped node located at the bottom-right corner in Figure 5.4. For each test case, we perform the same set of experiments 10 times and take the average over the results. We also show 95% confidence intervals in the figures reporting the results.

**Scenario 1—No Jamming**

We first compare the performance of JR-Seluge and Seluge when there is no jamming. To see the impact of code image size on performance, we use four different sizes: 10K bytes, 20K bytes, 30K bytes, and 40K bytes. Figures 4.3a and 4.3b show the completion time and communication overhead in the experiment, respectively.

![Figure 4.3](image-url)  
(a): Completion time in Scenario 1 (Sec)  
(b): Number of message transmissions in Scenario 1

Figure 4.3: Completion time and communication overheads of JR-Seluge and Seluge in scenario 1

Both JR-Seluge and Seluge show almost linearly increasing completion time and communi-
cation overhead as the image size increases. The completion time of JR-Seluge is about 74% to 113% longer than that of Seluge. Over all the experiments, the overhead for request and data packets of JR-Seluge is about 91% larger than that of Seluge, and JR-Seluge introduces about 74% more advertisement packets than Seluge.

These results indicate that JR-Seluge degrades the performance of Seluge where there is no jamming. However, the next evaluation will show that JR-Seluge can effectively survive jamming attacks that completely disable Seluge.

**Scenario 2—Jamming a Single Channel**

In this scenario, we compare Seluge and JR-Seluge when a single channel is under jamming attacks. We use four nodes (nodes 21, 24, 31, and 63) close to line 1 in Figure 5.4 as jammers, which divides our testbed into two halves. All jammers jam channel 26. We use 30K code image. We first run Seluge for one hour to see how far the dissemination flow can reach. As expected, Seluge’s dissemination flow is completely blocked by the jammers. Only the nodes to the right of line 2 in Figure 5.4 finish receiving the code image, but no nodes to the left of line 2 receive even a single page.

In contrast, JR-Seluge successfully bypasses the jammers; every node in the testbed completely receives the entire code image. Table 4.1 shows the performance overhead of JR-Seluge in this scenario. Compared with the case of no jammers, JR-Seluge introduces about 20% more delay, 3% more request and data packets, and 9% more advertisements.

Table 4.1: Completion Time and Communication Overhead of JR-Seluge (Seluge is unable to disseminate beyond line 2 in Figure 5.4.)

<table>
<thead>
<tr>
<th>JR-Seluge</th>
<th>Completion time (Sec)</th>
<th>Number of request and data packets sent</th>
<th>Number of advertisements sent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>190.95</td>
<td>15,139.83</td>
<td>7,121.5</td>
</tr>
</tbody>
</table>

**Scenario 3—Jamming Multiple Channels**

In this scenario, we examine the performance of JR-Seluge when multiple channels are jammed in a region of the network. We place 2 to 8 jammers close to line 1 in Figure 5.4, and perform experiments in four cases: (1) nodes 21 and 24 as 2 jammers; (2) nodes 21, 24, 31 and 63 as 4 jammers; (3) nodes 21, 24, 31, 63, 20, and 25 as 6 jammers; and (4) nodes 21, 24, 31, 63, 20,
25, 22, and 30 as 8 jammers. Different from Scenario 2, each jammer jams a different channel ranging from channel \((26 - \#jammers + 1)\) to 26. We use 30K code image in the experiments.

![Graph](image)

(a): Completion time in Scenario 3 and 4 (Sec)  
(b): Number of message transmissions in Scenario 3 and 4

Figure 4.4: Completion time and communication overheads of JR-Seluge in scenarios 3 and 4

JR-Seluge-Scenario3 in Figures 4.4a and 4.4b show the completion time, the number of advertisement packets, and the number of request and data packets sent in all the experiments. In both figures, JR-Seluge shows just a little increased overhead compared with the case of no jammers. In all cases, JR-Seluge introduces about 13% longer completion time, 4% more request and data packets, and 7% more advertisement packets than the case of no jammers.

**Scenario 4—Jamming Multiple Channels in Multiple Regions**

In this scenario, we evaluate the performance of JR-Seluge when multiple channels in multiple regions are under jamming. We scatter 2 to 8 jammers in various regions of the entire testbed, and perform the experiments in the following four cases: (1) nodes 13 and 50 as 2 jammers; (2) nodes 13, 50, 3, and 48 as 4 jammers; (3) nodes 13, 50, 3, 48, 24, and 61 as 6 jammers; and (4) nodes 13, 50, 3, 48, 24, 61, 35, and 66 as 8 jammers. Every jammer jams a different channel ranging from channel \((26 - \#jammers + 1)\) to 26. As in Scenario 3, we use the 30K code image.

JR-Seluge-Scenario4 in Figures 4.4a and 4.4b show the completion time and communication overhead in these experiments. As in scenario 3, compared with the case of no jammers, JR-Seluge shows slightly increased completion time and almost the same communication overhead. Over all cases, JR-Seluge shows on average 19% longer completion time, 5% more request and
data packets, and 4% more advertisement packets than the case of no jammers.

Based on the experimental evaluation, we can conclude that our channel migration scheme can effectively mitigate jamming threats. In contrast, Seluge fails when its dissemination channel is jammed. However, when there is no jamming attack, our channel migration scheme does introduce higher overhead than Seluge.

4.4 Summary

In this chapter, we presented a channel migration scheme to mitigate wireless jamming attacks. By exploiting the multiple wireless channels typically available on most wireless platforms, our scheme uses a flexible and resilient approach to switch communication channels, which enables wireless nodes to continue communication with their neighbors in the presence of jamming attacks. A nice property of the proposed scheme is that it does not depend on any single, fixed channel and executes in a decentralized and independent way on each wireless node. To investigate the performance of channel migration in real applications, we applied our channel migration scheme in a case study to enhance the jamming resiliency of Seluge. We evaluated it through both theoretical analysis and experimental evaluation. Both results demonstrated the capability of our channel migration technique in mitigating jamming attacks.
Chapter 5


In Chapter 3, we proposed Seluge which provides immediate authentication of code dissemination packets and resistance to DoS attacks aimed at the dissemination protocol. Besides such strong security protections, Seluge relies on ARQ (Automatic Repeat reQuest) for reliable dissemination of code images, with which each receiving node requests for retransmission of missing packets until it completely receives all of them. However, this ARQ mechanism is inefficient in lossy broadcast environments, which are typical in code dissemination, due to asymmetric packet loss between neighbor nodes.

Erasure codes – a forward error correction (FEC) technique – are a well-known approach for reliable communication. Several researchers demonstrated recently that using erasure codes along with ARQ can provide efficient and reliable code dissemination (e.g., [28, 22, 21, 53]). Based on these techniques, Bohli et al. integrated both integrity protection and erasure coding into code dissemination [7]. However, this approach does not provide immediate authentication of erasure-encoded packets, and consequently is vulnerable to DoS attacks that exploit authentication delays.

To address the lack of immediate authentication in erasure coding based code dissemination, LR-Seluge [69] was recently developed to securely integrate erasure codes into Seluge [26]. LR-Seluge offers better properties than [7]. Unfortunately, it does not take full advantage of erasure codes due to the lack of proactive transmission. As a result, it does not effectively reduce...
dissemination delays. In addition, the form of encoding and authenticating code images also prevents flexible application of erasure codes in LR-Seluge.

In this chapter, we present several new techniques to address the above problems and develop an efficient, reliable, and secure code dissemination scheme called FEC-Seluge. At the heart of FEC-Seluge is a novel proactive transmission mechanism, which achieves both efficiency and loss-tolerance through proactively transmitting redundant packets to enable receivers to tolerate packet losses.

A key parameter of the proactive transmission mechanism is the redundant rate, which determines the number of redundant packets. It is critical to choose an appropriate redundancy rate in FEC-Seluge to both meet the receivers’ decoding needs and reduce the bandwidth consumption. In this chapter, we also develop a method for estimating the optimal redundancy rate based on the observation of the current link condition between neighbor nodes, assuming that the packet loss rate is relatively stable during a short period of time.

A minor contribution in FEC-Seluge is a better method for encoding and authenticating code images. Though our method appears to be similar to LR-Seluge, the difference between FEC-Seluge and LR-Seluge allows flexible use of erasure codes in FEC-Seluge. In other words, FEC-Seluge can use any redundancy rate in packet encoding, while LR-Seluge is constrained with certain discrete values.

We have implemented FEC-Seluge as an extension to Seluge on TinyOS, and performed extensive empirical studies in a testbed network of 72 MicaZ motes. The experimental results show that FEC-Seluge outperforms LR-Seluge by about 30% and 23% in terms of dissemination delay and the number of packet transmissions, respectively.

The rest of this chapter is organized as follows. Section 5.1 clarifies our assumptions and threat model. Section 5.2 provides some preliminary information. Section 5.3 discusses the limitations of LR-Seluge, which motivate the design of FEC-Seluge. Section 5.4 presents the proposed techniques in FEC-Seluge. Section 5.5 describes the experimental evaluation of our approach in a wireless sensor network testbed. Section 5.6 summarizes this chapter.

5.1 Assumptions and Threat Model

**Assumptions:** We assume the source of the code images, i.e., the base station, is powerful and secure. The base station has a private/public key pair, and each node in the wireless sensor network is pre-configured with the base station’s public key. We assume that sensor nodes are resource constrained. A sensor node can perform a limited number of public key cryptographic operations, but cannot afford performing many such operations due to limited battery power. We assume that sensor nodes have enough memory to store the disseminated code image (e.g., using the measurement flash). We also assume that the base station can query the sensor nodes
in the network and gather their responses.

**Threat Model:** We assume the adversary has access to computationally resourceful nodes such as laptops. The adversary may launch both external and insider attacks. In external attacks, the adversary does not control any valid node in the network, but may attempt to eavesdrop for sensitive information, inject forged messages, and replay previously intercepted messages. Moreover, the adversary may launch wormhole attack [23], Sybil attacks [46], and Denial of Service (DoS) attacks. The adversary may also jam the communication channel; however, the adversary cannot constantly jam the communication channel without being detected and removed. In insider attacks, the adversary can compromise some nodes and attack the rest of the network using the sensitive information from the compromised nodes. The adversary may also control the compromised nodes not to cooperate with others (e.g., inject false data, drop packets). However, we assume that the majority of the nodes in the wireless sensor network are not compromised, and the adversary cannot compromise the base station.

5.2 Preliminaries

5.2.1 Erasure Codes

An erasure code is a forward error correction code. An erasure code consists of encoding and decoding operations. The encoding operation takes \( m \) equal-length symbols as input, and generates \( n \) encoded symbols, each of which has the same length as an input symbol (\( m \leq n \)). Given any \( m' \) out of \( n \) encoded symbols, the decoding operation reconstructs the \( m \) input symbols (\( m' \geq m \)). When \( m' = m \), the erasure codes are referred to as optimal codes. The ratio \( \frac{n-m'}{n} \) is called the redundancy rate. Some erasure codes are systematic codes, in which the original inputs symbols are embedded in the encoded output. In a systematic code, the output symbols besides the original input symbols are called redundant symbols.

5.2.2 LR-Seluge

LR-Seluge extends Seluge by integrating erasure codes for loss tolerance, considering each packet as a coding symbol [69]. In LR-Seluge, each page is encoded by adding several redundant packets. To authenticate every encoded packet in each page, LR-Seluge computes the hash images of all the encoded packets, splits them into equal-length pieces, and includes each piece in the corresponding original packet of the previous page before encoding. (Note that this design forces the recomputation of the encoded packets if the redundancy rate changes.) For the first page, LR-Seluge encodes the hash images of the encoded packets, and constructs a Merkle hash tree on top of the encoded results. Thus, after receiving enough number of encoded packets, a sensor node can recover the hash images of all encoded packets of the next
page, and immediately authenticate them if they are received.

To reduce the communication overhead due to the redundant packets, LR-Seluge includes an algorithm for a sender to decide a set of packets to broadcast when it receives requests for a code page from its neighbors (receivers). The key idea is to select the packet desired by more receivers with higher priority and eventually broadcast the minimum set of packets that can satisfy all the receivers’ needs of additional packets required for decoding. However, this strategy prevents LR-Seluge from taking full advantage of erasure codes. The fundamental reason is the lack of proactive transmission for some receivers. As a result, LR-Seluge does not effectively reduce dissemination delays.

5.3 Limitations of LR-Seluge

To see the limitations of LR-Seluge, we first discuss its packet transmission algorithm in further detail. In LR-Seluge, once a node \(r\) detects a neighbor \(s\) who has a code page it needs to download, it sends a request message to \(s\), which includes a packet bit vector to specify the packets that \(r\) has not received yet on that page. Dissemination of the page is done through a sequence of rounds. In each round \(r\) sends a request to \(s\), and \(s\) broadcasts a set of packets for the request. The packet transmission algorithm in LR-Seluge is to decide the set of broadcast packets for given requests (bit vectors).

The algorithm refers to two factors so called popularity and distance. The popularity of a packet is the number of receivers who request the packet. The distance of a receiver means the minimum number of additional packets the receiver requires for decoding the page. Based on the received packet bit vectors, \(s\) calculates the popularity of each packet, and then broadcasts the first packet with the highest popularity. After that, \(s\) clears all 1 bits corresponding to the broadcast packet in the received bit vectors, and reduces by 1 the distance value of each receiver whose 1 bit was just cleared. Once the distance of a receiver reaches 0, \(s\) deletes the receiver’s bit vector from its memory. As long as \(s\) has any remaining bit vectors in its memory, it repeats the above procedure to identify the next packet to broadcast.

Now let us discuss the limitation of the LR-Seluge packet transmission algorithm. To effectively demonstrate the problem, we use the following example scenario. Figures 5.1a to 5.1d show the packet bit vectors received by \(s\) in each round, when receivers \(r_1\), \(r_2\), and \(r_3\) download a single code page from \(s\) through 4 rounds. We assume that the page consists of 6 original and 6 redundant packets. In the figures, each row represents the bit vector from the corresponding receiver, and each bit “1” indicates that the receiver has not received the corresponding packet. The packets marked with “X” in the figures represent that they are missed by the receiver in that round. Column “d” shows the distance of each receiver.

According to the algorithm, in round 1 \(s\) broadcasts packets 0 to 5 (marked with the bold
Figure 5.1: An example sequence of rounds in LR-Seluge

line). If any receiver misses a single packet, it will require another round of transmission. In round 2, $s$ broadcasts packets 6 to 8. If $r_3$ misses any one of them, it will require an additional round. The same problem appears in rounds 3.

In general, LR-Seluge provides no proactive transmission in each round. Thus, for the receiver (e.g., $r_3$) who requires the most packets, even a single lost packet will cost another round. In contrast, if the sender proactively transmits extra packets already encoded by erasure coding, it is likely that the above receiver will not need an additional round. This example shows that LR-Seluge does not fully take advantages of erasure codes. Moreover, in LR-Seluge, besides the receiver that requires the most packets in a round (e.g., $r_3$), the remaining receivers (e.g., $r_1$ and $r_2$) have to suppress their requests for the next page until that receiver (e.g., $r_3$) finishes downloading the current page. This feature further introduces additional delays to the code dissemination process.

Figure 5.1c also demonstrates another problem in the packet transmission algorithm. According to the algorithm, $s$ broadcasts packets 4 and 5 in round 3. However, packet 6 is a better choice than packet 5, because broadcasting packet 6 benefits both $r_2$ and $r_3$. The reason for this problem is that after broadcasting packet 4, $s$ deletes $r_2$’s bit vector from its memory because the distance of $r_2$ reaches 0. Thus when choosing another packet for $r_3$, $s$ no longer considers the packets desired by $r_2$, and eventually broadcasts packet 5 rather than packet 6.

5.4 Design of FEC-Seluge

In this section, we present the design of FEC-Seluge, which is an extension to Seluge [26]. Our main contribution in FEC-Seluge is a novel proactive transmission mechanism to address the inefficiency due to the lack of proactive transmission in LR-Seluge. The key idea is to
proactively transmit encoded, redundant packets so that receivers can tolerate certain packet losses. In addition, we propose a slightly different encoding scheme from LR-Seluge. Even though the encoding procedure in FEC-Seluge appears to be similar to LR-Seluge, our scheme allows flexible application of erasure codes. In other words, FEC-Seluge can use any redundancy rate in packet encoding, while LR-Seluge is constrained with certain discrete values.

5.4.1 Encoding and Authenticating a Code Image

For the sake of presentation, we use the following notations: $h(\cdot)$ denotes a cryptographic hash function; $C \parallel D$ denotes the concatenation of $C$ and $D$; $|E|$ denotes the size of $E$ in byte; $m$ denotes the number of original packets in each page; $R$ denotes the redundancy rate for erasure coding; and $n \left(= \left\lceil \frac{m}{1-R} \right\rceil \right)$ denotes the total number of encoded packets in each page after encoding $m$ packets with redundancy rate $R$.

Given a code image for dissemination, the base station first divides it into $P$ chunks. Let $C_i$ denote the $i$th chunk ($1 \leq i \leq P$). The length of each chunk from $C_1$ to $C_{P-1}$ is $m|\text{payload}| - n|h(\cdot)|$ bytes, while $|C_P|$ is $m|\text{payload}|$ bytes ($|\text{payload}|$ represents the size of a packet payload). $C_i$ is assigned to the $i$th page. Starting from $C_P$, the base station processes each chunk in reverse order to generate the packets of the corresponding page.

Figure 5.2 illustrates this process. The base station splits $C_P$ into $m$ payloads, and then runs the encoding operation with them, which results in $n$ encoded payloads of $\text{pkt}_i$ ($1 \leq i \leq n$) on page $P$. The base station then computes the hash images of all $n$ packets (including the packet headers) and concatenate them together.

In LR-Seluge the concatenated hash images are first split into $m$ equal-length pieces, and then each piece is included in the corresponding packet of the previous page before encoding [69]. Thus only the redundancy rates with which the length of the resulting hash image is a multiple of $m$ can be used in LR-Seluge. If other redundancy rates are used, LR-Seluge either wastes some of packet payloads or has to adjust the parameters $m$ and $n$. To address this problem, in FEC-Seluge the base station appends the concatenated hash images to $C_{P-1}$, which extends $|C_{P-1}|$ to $m|\text{payload}|$ bytes. In this way, any redundancy rate can be used by adjusting the initial length of chunks $C_1$ to $C_{P-1}$ accordingly.

After processing chunk $C_P$, the base station repeats the same procedure to chunks $C_{P-1}$, $C_{P-2}$, ..., and $C_1$.

By following the same procedures as in LR-Seluge [69], the base station encodes the hash images of every encoded packet in page 1, constructs a Merkle hash tree using the encoded results, and then generates all page 0 packets from the hash tree. Finally, the base station constructs a signature packet, which includes the root of the Merkle hash tree, the meta data about the code image (e.g., version number, size), and a signature over all of them. To achieve
the immediate authentication property, the base station disseminates the packets starting from the signature packet and then the code pages in a sequential order from page 0 to page $P$.

5.4.2 Proactive Packet Transmission

This section presents the proposed packet transmission algorithm in FEC-Seluge. As mentioned earlier, our packet transmission mechanism achieves both efficiency and loss-tolerance through proactively transmitting redundant packets to enable receivers to tolerate packet losses in decoding.

We assume that sender $s$ has received requests for the same page from $R$ different receivers ($r_i, 1 \leq i \leq R$). With the packet bit vectors in the received request messages, $s$ can figure out the set of packets of the page requested by each receiver. Let $S_i$ denote the set corresponding to receiver $r_i$. In our algorithm, if any of the received requests indicates that the corresponding receiver has never received any packet of the page yet, $s$ broadcasts all $n$ packets. Otherwise,
Algorithm 1:

1: $X = \{\}$
2: For each packet in $\bigcup_{i=1}^{R} S_i$, count the number of receivers who have not received the packet yet.
3: for $i = 1$ to $R$ do
4:   if $|X \cap S_i| < y_i$ then
5:     Sort the packets in $S_i - X$ according to their popularity values.
6:     Append the $y_i - |X \cap S_i|$ highest popularity packets in $S_i - X$ to $X$.
7:   end if
8: end for

$s$ estimates the packet loss rate each receiver has experienced in the previous round. Based on the estimated packet loss rates, $s$ decides how many and what packets it will broadcast in this round.

To estimate the packet loss rate, after sending a request to $s$, $r_i$ counts the number of packets ($u_i$) it received from $s$ in this round. $s$ also counts the number of packets ($v$) it broadcasts in this round. When sending a request message for retransmission to $s$ in the next round, $r_i$ piggybacks $u_i$ to the message to notify $s$. If $s$ receives the request, $s$ can calculate the packet loss rate ($p'_i = \frac{v - u_i}{v}$) for $r_i$ in the previous round. Then $s$ guarantees that the final packet transmission set of the current round includes at least $y_i$ ($= \min([\frac{n - (n - |S_i|)}{1 - p'_i}], |S_i|)$) packets among all the packets in $S_i$. Intuitively, this is to transmit enough packets so that $r_i$ can decode even under the packet loss rate of $p'_i$.

Algorithm 1 describes the detailed procedure for $s$ to derive the set of packets ($X$) it will broadcast for $r_1, r_2, \ldots, r_R$. To avoid the problem in LR-Seluge (Figure 5.1c), $s$ constructs the packet popularity table and refers to it over the packet selection process. For each packet in $\bigcup_{i=1}^{R} S_i$, the table shows how many receivers have not received the packet yet. For each receiver $r_i$, $s$ repeats lines 4 to 7. In line 4 $s$ checks if $X$ already includes enough number of packets for $r_i$. Only if the answer is “no”, $s$ adds more packets to $X$ from $S_i - X$, where the added packets are the $y_i - |X \cap S_i|$ most popular packets in $S_i - X$.

![Figure 5.3: An example sequence of rounds in our approach](image)
Figures 5.3a and 5.3b show the packet bit vectors received by $s$ in each round, when $r_1$, $r_2$ and $r_3$ download a single code page consisting of 6 original and 2 redundant packets from $s$ in FEC-Seluge. In round 1, $s$ broadcasts all 8 packets, proactively transmitting 2 extra packets (packets 6 and 7). Even if a receiver misses some original packets in round 1, it can still recover them through decoding if it receives at least 6 out of 8 packets. We assume that each receiver has missed the packets marked with “X” in round 1. At the end of round 1, $r_2$ still requires one more packet for decoding, and the packet loss rate for $r_2$ in round 1 is $3/8$. Thus for $r_2$, our algorithm guarantees that the packet transmission set $X$ includes at least 2 packets among packets 1, 5, and 7 missed by $r_2$ in round 1. Since packets 1 and 7 have higher popularity than packet 5, $s$ adds packets 1 and 7 to set $X$. Similarly for $r_3$, $X$ should include at least 4 packets among packets 0, 1, 4, and 7, which are missed by $r_3$ in round 1. Since $X$ already includes packets 1 and 7, $s$ additionally puts packets 0 and 4 into $X$. Finally, $s$ broadcasts packets 0, 1, 4, and 7. This broadcast provides proactive transmission of 1 extra packet to $r_2$ and 2 extra packets to $r_3$, and consequently $r_2$ and $r_3$ finish downloading this page despite some packet loss in round 2.

5.4.3 Optimal Redundancy Rate

During the use of proactive transmission, a key parameter is the redundant rate, which determines the number of redundant packets that need to be transmitted. It is critical to choose an appropriate redundancy rate in FEC-Seluge to both meet the receivers’ decoding needs and reduce the bandwidth consumption. In the following, we develop a technique for estimating the optimal redundancy rate based on the observation of the current link condition between neighbor nodes.

We have each sensor node periodically measure the packet loss rates of all the down links from its neighbors to itself, by counting the number of packets it has received among those each neighbor has transmitted for a certain period of time. When needed (i.e., the need of disseminating a new code image), the base station distributes a query over the entire network. Each node then calculates the expected packet loss rate when receiving packets from its neighbors based on the latest measurements, and replies to the base station. By analyzing the received reports, the base station can decide the redundancy rate it will use for erasure coding.

Expected Packet Loss Rate

Now let us discuss how each node calculates the expected packet loss rate when it has the measured packet loss rates of all the links from its neighbors. Let $r$ represent any single node in the network, and $r$ has $n$ neighbors ($s_i$, $1 \leq i \leq n$) who have a code page it needs to download. $p_i$ denotes the packet loss rate of the down link from $s_i$. Let $X$ be a random variable to represent
the packet loss rate for \( r \) when a neighbor broadcasts encoded packets. In FEC-Seluge, every node periodically broadcasts an advertisement. If \( r \) first receives \( s_i \)'s advertisement among all the neighbors' for a certain advertisement period, it selects \( s_i \) as sender and transmits a request for the page to \( s_i \). \( s_i \) broadcasts packets after it receives the request. Thus \( X = p_i \) in this case. If \( r \) receives a query message from the base station, it replies with \( E[X] \), the expected packet loss rate \( r \) experiences when receiving packets from the neighbors.

To compute \( E[X] \), we assume that in each time slot every neighbor broadcasts an advertisement at a random time point, and \( r \) receives at least one of them. With this assumption, \( r \) may receive multiple advertisements in a time slot, and there are \( \sum_{i=1}^{n} \binom{n}{i} \) different cases in total. Considering all these cases makes the computation of \( Pr(X = p_i) \) very expensive, and this is unacceptable for resource constrained sensor devices. Thus we additionally assume that there are only the cases in which \( r \) receives exactly one advertisement in each time slot. We also assume there is no loss of request packets. Then \( Pr(X = p_i) = \frac{(1 - p_i) \prod_{k \neq i}^{n} p_k}{\sum_{j=1}^{n} (1 - p_j) \prod_{k \neq j}^{n} p_k} \) and

\[
E[X] = \sum_{i=1}^{n} p_i Pr(X = p_i).
\]

**Optimal Redundancy Rate**

Assume that the base station has received the expected packet loss rates from every network node. Given that each code page consists of \( \lceil \frac{m}{1 - \text{RedRate}} \rceil \) encoded packets in total, the base station calculates the number of rounds and encoded packet transmissions required for each node \( r \) to receive at least \( m \) packets of the page by following Algorithm 2, and takes the averages over every node. In Algorithm 2, we assume that \( r \) misses each packet broadcast by a neighbor node with the probability of \( x_r \), which is the expected loss rate reported by \( r \). By repeating the same calculation for different redundancy rates, the base station finds out the optimal redundancy rate that allows the maximum performance gain compared to the case of no redundancy (\( \text{RedRate} = 0 \)). To get more reliable estimation, the base station can run the procedure several times and take the average. The resulting average can be used as the optimal redundancy rate in FEC-Seluge.

**5.4.4 Security Analysis**

The following malicious attacks are possible against our estimation method. First, the adversary may inject fake query messages so that the entire network keeps busy for replying to the base station. To prevent such attacks, the base station needs to sign query messages with its private key. Second, due to the use of signatures, the adversary may launch DoS attacks by injecting bogus query messages with invalid signatures [47]. To cope with this problem, we assume that
Algorithm 2:

\begin{verbatim}
n = \lceil \frac{m}{1.0 - \text{RedRate}} \rceil, \ numRound = 0
\text{totalNumTxPkt} = 0, \text{totalNumRxPkt} = 0
\textbf{while} \ totalNumRxPkt < m \textbf{do}
  \hspace{1em} \textbf{if} numRound = 0 \textbf{then}
    \hspace{2em} \text{pktLossRate} = \text{RedRate}
  \hspace{1em} \textbf{else}
    \hspace{2em} \text{pktLossRate} = \frac{\text{numTxPkt} - \text{numRxPkt}}{\text{numTxPkt}}
  \textbf{end if}
  \hspace{1em} \textbf{if} \ pktLossRate = 1.0 \textbf{then}
    \hspace{2em} \text{numTxPkt} = n - \text{totalNumRxPkt}
  \hspace{1em} \textbf{else}
    \hspace{2em} \text{numTxPkt} = \min(\lceil \frac{m - \text{totalNumRxPkt}}{1.0 - \text{pktLossRate}} \rceil, n - \text{totalNumRxPkt})
  \textbf{end if}
  \hspace{1em} \text{totalNumTxPkt} = \text{totalNumTxPkt} + \text{numTxPkt}
  \hspace{1em} \text{numRxPkt} = 0
\textbf{while} \ \text{numTxPkt} > 0 \textbf{do}
  \hspace{2em} \textbf{if} \not\text{(drop this packet with } x_i) \textbf{then}
    \hspace{3em} \text{numRxPkt} + +
  \hspace{2em} \textbf{end if}
  \hspace{2em} \text{numTxPkt} - -
\textbf{end while}
\hspace{1em} \text{totalNumRxPkt} = \text{totalNumRxPkt} + \text{numRxPkt}
\hspace{1em} \text{numRound} + +
\textbf{end while}
\textbf{return} \ \text{numRound} \ \text{and} \ \text{totalNumTxPkt}
\end{verbatim}

Each query message is also protected with Message Specific Puzzle [47], which allows sensor nodes to effectively filter out bogus query messages without expensive signature verification.

Finally, the adversary may transmit bogus reports to the base station or forge reports received from legitimate nodes for further propagation. To deal with this problem, we assume that the base station and each sensor node share a common secret, which is used to authenticate a report from the sensor node. Consequently, external attackers cannot provide forged reports to the base station. However, this does not prevent compromised nodes from sending forged reports to the base station. To mitigate the impact of such malicious reports on the estimation, we can use robust statistical techniques, such as discarding some extreme values of the received expected loss rates. In the next section, we will present our experimental evaluation to see the impact of such bogus reports and the effectiveness of eliminating extreme values.

5.5 Experimental Evaluation

In this section, we report the experimental evaluation of FEC-Seluge in a wireless sensor network testbed. We have implemented FEC-Seluge as an extension to Seluge, where we used a systematic version of Reed-Solomon codes for erasure coding. For comparison purpose, we
have also implemented the packet transmission algorithm of LR-Seluge, and integrated it into our code base. In the following experiments, we have each code page include $m = 20$ original packets before encoding. All the remaining parameters in the experiments are the same as in [26].

In this evaluation, we first perform a set of experiments to empirically study the optimal redundancy rate in FEC-Seluge for our testbed network. Then based on the empirically obtained optimal redundancy rate, we test our redundancy rate estimation technique under two different attack scenarios. Finally, we compare the performance of FEC-Seluge, LR-Seluge, and Seluge.

**Testbed:** We perform the experiments in a sensor network testbed of 72 MicaZ motes. Figure 5.4 shows the layout of the testbed. The testbed area includes offices, labs, server rooms, and corridors, covering an area of 152.5 feet × 97 feet. We set the transmission power level of the MicaZ radio module (CC2420) as 0dBm. In each experiment, the dissemination starts from the source node in Figure 5.4.

**Evaluation Metrics:** We use two metrics in our evaluation: *Dissemination delay* and *communication overhead*. The dissemination delay is the time required to finish disseminating a code image to all the nodes in the network. As mentioned in [24], code dissemination requires that every node keep its radio on. Thus, the dissemination delay is closely related to the energy consumption required by code dissemination. The communication overhead is measured as the total number of packets transmitted by all the nodes during a code dissemination, which is also related to radio power consumption.

For each test case, we perform the same set of experiments 20 times and take the average over the results. We also show 95% confidence intervals in the figures reporting the results.
5.5.1 Optimal Redundancy Rate

This set of experiments is for empirically finding out the optimal redundancy rate in FEC-Seluge, which enables the best performance in our testbed network. For this objective, we run FEC-Seluge with different redundancy rates from 0.0 to 0.5 with 0.02 spacing. We disseminate $30K$ code image.

Figures 5.5a and 5.5b show the dissemination delay and communication overhead, respectively. The dissemination delay and communication overhead reach the minimum averages at the redundancy rates of 0.24 and 0.22, respectively. However, other redundancy rates around those two rates show the results very close to the minimum averages.

We further perform statistical analysis using one-way analysis of variance (ANOVA) [58] and Dunnett’s test [16] to identify all the redundancy rates whose results have no statistically meaningful difference from the minimum averages and consequently the range of the optimal redundancy rates. In particular, using ANOVA we confirm that there do exist statistically meaningful differences among the redundancy rates we have tested.

Using Dunnett’s test, we finally identify those redundancy rates that are not statistically different from the redundancy rates introducing the min averages. The analysis results show that there is no significant difference in terms of dissemination delay over the redundancy rates 0.14-0.36 and in terms of communication overhead over the redundancy rates 0.14-0.36 and 0.4. It suggests that the range of the optimal redundancy rates in both dissemination delay and communication overhead is from 0.14 to 0.36.

5.5.2 Defense against Bogus Reports

As mentioned earlier, compromised nodes can provide bogus reports to the base station to disrupt the estimation of redundancy rate. One possible attack strategy is to provide extremely high or low noise values. To investigate the impact of such attacks, we randomly pick extremely high values in the range of 0.9 to 1.0 (or in the range of 0.0 to 0.1 in a separate experiment) as bogus reports. However, this type of misbehavior may be easily detected. Thus, as another strategy, we assume that the adversary picks a random value between 0.0 and 1.0. Tables 5.1 and 5.2 show the results. Because the optimal redundancy rates for our testbed network is small values, the extremely low noise attacks showed almost no impact on our estimation. We omit the exact numbers due to space limit.

To see the impact of bogus report ratio, we run experiments as increasing the ratio of compromised nodes from 0% to 50%. For each ratio, we randomly choose that ratio of nodes among all the testbed nodes, and then replace the expected loss rates of those nodes with randomly picked values in a given attack scenario. We generate 10 test inputs in this way for each ratio, and run our estimation with each test input. In addition, to see the effectiveness
of dropping extreme values as the countermeasure, we also run the estimation after dropping a certain percentage of lower and upper values of the values in each test input. The column index of each table shows the dropping rate we use. For instance, 5% dropping rate means dropping the values in the lower 5% and the upper 5% in a given test input. The result in each test case shows the average of our estimations returned in each run of the case.

As shown in the tables, the estimated redundancy rates increase as the ratio of bogus reports increases in both the extremely high and random noise attack scenarios. Under the random
Table 5.1: Estimated Redundancy Rates under extreme high noise attacks.

<table>
<thead>
<tr>
<th>Percentage of Bogus Reports</th>
<th>Dropping Rate</th>
<th>0%</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
<th>25%</th>
<th>30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td></td>
<td>0.16</td>
<td>0.17</td>
<td>0.18</td>
<td>0.16</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>10%</td>
<td></td>
<td>0.32</td>
<td>0.22</td>
<td>0.18</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td>0.39</td>
<td>0.34</td>
<td>0.29</td>
<td>0.22</td>
<td>0.18</td>
<td>0.18</td>
<td>0.17</td>
</tr>
<tr>
<td>30%</td>
<td></td>
<td>0.62</td>
<td>0.45</td>
<td>0.40</td>
<td>0.35</td>
<td>0.31</td>
<td>0.24</td>
<td>0.18</td>
</tr>
<tr>
<td>40%</td>
<td></td>
<td>0.64</td>
<td>0.56</td>
<td>0.52</td>
<td>0.48</td>
<td>0.45</td>
<td>0.39</td>
<td>0.36</td>
</tr>
<tr>
<td>50%</td>
<td></td>
<td>0.66</td>
<td>0.61</td>
<td>0.58</td>
<td>0.53</td>
<td>0.53</td>
<td>0.51</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Table 5.2: Estimated Redundancy Rates under random noise attacks.

<table>
<thead>
<tr>
<th>Percentage of Bogus Reports</th>
<th>Dropping Rate</th>
<th>0%</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
<th>25%</th>
<th>30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td></td>
<td>0.16</td>
<td>0.17</td>
<td>0.18</td>
<td>0.16</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>10%</td>
<td></td>
<td>0.22</td>
<td>0.18</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
<td>0.16</td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td>0.26</td>
<td>0.19</td>
<td>0.18</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>30%</td>
<td></td>
<td>0.28</td>
<td>0.20</td>
<td>0.19</td>
<td>0.18</td>
<td>0.18</td>
<td>0.18</td>
<td>0.17</td>
</tr>
<tr>
<td>40%</td>
<td></td>
<td>0.30</td>
<td>0.23</td>
<td>0.22</td>
<td>0.21</td>
<td>0.20</td>
<td>0.19</td>
<td>0.18</td>
</tr>
<tr>
<td>50%</td>
<td></td>
<td>0.33</td>
<td>0.25</td>
<td>0.24</td>
<td>0.22</td>
<td>0.22</td>
<td>0.21</td>
<td>0.21</td>
</tr>
</tbody>
</table>

noise attacks, our estimations in every test case falls into the optimal range, which shows the robustness of our method against this type of attack. On the other hand, under the extremely high noise attacks, our estimation method only returns the optimal redundancy rate in the case of 10% of bogus reports, while it fails to hit the optimal range in the cases of 20% to 50% of bogus reports. But in the cases of 20%, 30%, and 40% of bogus reports, our method can reach the optimal range by dropping 5%, 15%, and 30%, respectively.

5.5.3 Comparison of FEC-Seluge, LR-Seluge, and Seluge

We perform another set of experiments to compare FEC-Seluge with LR-Seluge and Seluge. To see the impact of code image size on performance, we use four different code images (10K, 20K, 30K, and 40K). To confirm the problem due to the lack of proactive transmission in LR-Seluge, we slightly change the packet transmission algorithm of LR-Seluge so that a sender broadcasts all $n$ packets of a page when it has received a request, which indicates the receiver has not received any packet of the page yet, and compare this variation with LR-Seluge. We refer to this as a variation of LR-Seluge. We use two redundancy rates of 0.16 and 0.24. The rate 0.16 is the redundancy rate our estimation method returns for the testbed network, and 0.24 is the redundancy rate which shows the overall best performance in the experiments in Section 5.5.1.
Figures 5.6a to 5.6d show the dissemination delay and the number of advertisement, request, and encoded packet transmissions, respectively.

We first compare FEC-Seluge with LR-Seluge. As indicated in the figures, FEC-Seluge shows much better performance than LR-Seluge in dissemination delay and the number of advertisement and request packet transmissions. Over all the test cases, FEC-Seluge requires on average 29% shorter delay, 30% less advertisements, and 40% less request packet transmissions than LR-Seluge. All these results confirm the effectiveness of our proactive transmission mechanism. Moreover, despite the use of proactive transmission, FEC-Seluge even shows on average 9% less encoded packet transmissions than LR-Seluge. The reason for this result is that our proactive transmission mechanism increases the chance for all the receivers in a neighborhood to finish downloading a page around the same time, and consequently makes all of them syn-
chronized in terms of the page they are trying to download. This eventually minimizes the redundant transmissions of the same page in the same neighborhood due to the lack of page synchronization among the neighbors. Compared with Seluge, FEC-Seluge shows on average 35% shorter delay and 28% less overall packet transmissions.

On the other hand, LR-Seluge shows only fractional performance improvements over Seluge. In particular, the dissemination delay of LR-Seluge is on average 9% shorter than that of Seluge. In the number of overall packet transmissions, LR-Seluge shows about 6% better performance than Seluge. To confirm that the reason for such poor performance of LR-Seluge is the lack of proactive transmission, we compare LR-Seluge with the variation of LR-Seluge. As expected, the variation shows about 16% shorter delay, 19% less advertisements, 29% less request packets than LR-Seluge.

Finally, FEC-Seluge with 0.16 and 0.24 shows similar performance. The performance overhead in 0.16 is on average 4% larger than that in 0.24.

5.6 Summary

In this chapter, we proposed a new secure code dissemination approach named FEC-Seluge, which is aimed at improving the performance by using forward error correction. With the novel proactive transmission mechanism, FEC-Seluge effectively reduces both the dissemination delay and the communication overhead. To balance the efficiency and loss-tolerance capabilities, we developed a technique to estimate the optimal amount of redundancy desired for a given network condition. The empirical study using our testbed network showed that FEC-Seluge significantly outperforms the previous approaches.
Chapter 6

Conclusion and Future Work

6.1 Contributions

In this thesis, we addressed the following problems in order to provide efficient, secure, and reliable code dissemination in wireless sensor networks: (1) integrity protection of disseminated code images and immunity from DoS attacks exploiting code dissemination protocols, (2) jamming-resilient code dissemination, (3) loss-tolerant and secure code dissemination using erasure codes.

1. integrity protection of disseminated code images and immunity from DoS attacks exploiting code dissemination protocols: Several code dissemination protocols [13, 24, 57, 30, 50, 29, 52, 44] were developed to propagate new code images using the ad-hoc wireless network formed by the sensor nodes in a network. However, all of them are vulnerable to malicious attacks due to no consideration of security in their design.

Several researchers developed secure code dissemination protocols [17, 14, 31] as an extension to Deluge [24], the most widely used as a de facto code dissemination system in TinyOS. However, their mechanisms to authenticate code images are either vulnerable to DoS attacks due to authentication delays or disable the efficient page-by-page dissemination in Deluge as preventing the seamless integration into Deluge. Moreover, all these approaches overlooked DoS attacks exploiting protocol control messages and expensive signature verifications.

To address these limitations, in Chapter 3 we proposed Seluge, which is also a secure extension to Deluge. Compared with the previous attempts [17, 31, 14] for secure code dissemination, Seluge not only provides integrity protection for disseminated code images, but is also resistant to various DoS attacks exploiting the expensive signature verification operations, possible authentication delays, and the efficient epidemic propagation strate-
gies used by Deluge. Through the security and performance analysis, we demonstrated that Seluge is superior to all the previous solutions [17, 31, 14]. In addition, we integrated the proposed techniques into the Deluge code base in TinyOS distributions, providing a software packet that is readily available, and using the implementation we performed extensive experiments in a network of MicaZ motes which showed the efficiency of Seluge in practice.

2. jamming-resilient code dissemination: Despite the security protections in Seluge, it is still vulnerable to wireless jamming attacks; if an attacker jams the communication channel for code dissemination, Seluge would be unable to complete code dissemination. As discussed in Chapter 2.2, several groups of researchers developed anti-jamming techniques based on either physical layer spread spectrum or upper layer channel hopping (e.g., [54, 64, 45, 67, 20, 66, 59, 60, 55, 51, 39, 35, 36, 32]). However, all these existing solutions have some limitations as follows, which make them infeasible for jamming resilient code dissemination in wireless sensor networks. Some of the existing approaches (e.g., [54, 64, 45, 67, 20, 66]) can be easily defeated by insider jammers due to their dependency on a pre-shared secret between communicating parties. The recent spread spectrum techniques (e.g., [59, 60, 55, 51, 39]) which addressed this dependency problem all require changes to the physical radio devices, which are not available on the current wireless sensor platforms. Finally, the existing channel hopping approach which does not require a pre-shared secret can only use a small fraction of the available bandwidth for the purpose of anti-jamming, and consequently increases overhead significantly.

To provide jamming resilience property to Seluge by addressing the above-mentioned limitations, we proposed a new approach called channel migration to mitigating wireless jamming attacks. By exploiting the multiple wireless channels typically available on existing wireless sensor platforms, our scheme uses a flexible and resilient approach to switch communication channels, which enables sensor nodes to continue the normal code dissemination operations in the presence of jamming attacks. A nice property of the proposed scheme is that it does not depend on any single, fixed channel and executes in a decentralized and independent way on each wireless node. In addition, to improve channel synchronization between neighbors, we developed an advertisement mechanism, which enables each node to periodically inform its neighbors using different communication channels about its current main communication channel. To enhance the jamming resilience of Seluge and investigate the effectiveness of the proposed channel migration scheme, we integrated the channel migration scheme into Seluge and evaluated the resulting protocol through both theoretical analysis and experimental evaluation in a testbed of MicaZ motes. Both results indicated that our channel migration scheme can effectively
and efficiently mitigate jamming attacks.

3. **loss-tolerant and secure code dissemination using erasure codes**: Seluge relies on ARQ for reliable dissemination of code images, which is inefficient in lossy broadcast environments. Several researchers demonstrated recently that using erasure codes along with ARQ can provide efficient and reliable code dissemination (e.g., [22, 21, 53]). Based on these techniques, Bohli et al. integrated both integrity protection and erasure coding into code dissemination [7]. However, this approach does not provide immediate authentication of erasure-encoded packets, and consequently is vulnerable to DoS attacks that exploit authentication delays.

To address the lack of immediate authentication in erasure coding based code dissemination, LR-Seluge [69] was recently developed to securely integrate erasure codes into Seluge. However, due to the lack of proactive transmission, LR-Seluge does not take full advantage of erasure codes. As a result, it does not effectively reduce dissemination delays. In addition, the form of encoding and authenticating code images also prevents flexible application of erasure codes in LR-Seluge.

In Chapter 5, we proposed a new approach called **FEC-Seluge**, which is aimed at improving the code dissemination performance by better benefiting from forward error correction while preserving the same level of security protections as in Seluge. At the heart of FEC-Seluge is a novel *proactive* transmission mechanism, which achieves both efficiency and loss-tolerance in code dissemination. To balance the efficiency and loss-tolerance capabilities, we developed a technique to estimate the optimal amount of redundancy in proactive transmission for a given network condition. We implemented and evaluated FEC-Seluge through experiments in a testbed environment. The experimental results showed that FEC-Seluge significantly outperforms both Seluge and LR-Seluge.

### 6.2 Future Work

Although our research in this thesis has accomplished many advances for efficient, secure, and reliable code dissemination service in wireless sensor networks, there still remain several problems to be addressed.

1. **Techniques to prevent and detect insider attacks**: Due to the open and unattended nature of wireless sensor networks, it is easy for an adversary to capture legitimate sensor nodes and retrieve some useful information (e.g., keys for message authentication) for her attacks from the captured nodes. Despite the proposed security mechanisms, two types of insider attacks using such compromised nodes are still possible against our approaches.
Firstly, insider attackers may introduce fake control messages with valid authenticators to disable the efficient dissemination mechanisms based on control messages or waste the limited power resources of victim nodes. A possible countermeasure is to check behavioral integrity of each node to identify compromised nodes being misused for attacks. Secondly, insider attackers may provide false reports to the base station in FEC-Seluge to disable the accurate estimation of the optimal redundancy rate. To deal with such type of attacks, we need more investigation on robust statistical techniques to effectively filter out bogus reports or make the impact of false reports on the final estimation negligible.

As more fundamental solution, software attestation which detects illegal changes on the memory of sensor devices is a promising technique to identify compromised nodes. More research is required to develop an attestation mechanism which is suitable for resource-constrained sensor devices.

2. Techniques to preserve the secrecy of code images: Depending on the application of wireless sensor networks, the secrecy of code images may be very important. For instance, in military applications, it may be required to prevent enemies from identifying the code image being propagated over the network. Moreover, it may be necessary to protect the secrecy of code images even when the enemies capture the sensor nodes which are running the code images in this case. More research is required to achieve these objectives.

3. Anti-jamming techniques: As demonstrated in Chapter 4, our channel migration scheme requires much larger performance overhead than Seluge when there is no jamming. In addition, it cannot survive reactive jamming attacks in which the jammers quickly search through the available channels and jam the channels where there exist on-going transmissions. Thus in the future research, we will investigate on techniques which can offer better jamming resilience with lower performance overhead.
REFERENCES


