UNIVERSITY OF CALIFORNIA

Los Angeles

Revere—Disseminating Security Updates at Internet Scale

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Jun Li

2002

The dissertation of Jun Li is approved.

_____

Leonard Kleinrock

_____

Miodrag Potkonjak

_____

William Kaiser

_____

Gerald Popek

_____

Richard Muntz, Committee Co-chair

_____

Peter Reiher, Committee Co-chair

University of California, Los Angeles

2002

To my parents

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Figures

Tables

ACKNOWLEDGMENTS

Without the constant light of guidance, support and encouragement from my advisors, Peter Reiher and Gerald Popek, I would have never reached the culmination of this research journey. During my six years at the University of California, Los Angeles, I have had the great fortune to be their advisee. They tremendously helped to make a young, inexperienced graduate student into the researcher that I am today. I am certain that in the future I will continue to benefit from the guidance I have received. Their creation and continued advancement of the Laboratory for Advanced Systems Research (LASR)—a unique research laboratory with nearly 30 years of history—also afforded me a wonderful opportunity to learn, to grow, and to flourish.

I also owe a great magnitude of thanks to my doctoral committee. In addition to Peter and Gerald, I am honored to have Dr. Leonard Kleinrock, Dr. Miodrag Potkonjak, Dr. Richard Muntz and Dr. William Kaiser on my committee. Their sharp, insightful feedback, offered since the very beginning of this research, has proved invaluable.

My participation in other research projects also benefited my thesis research indirectly but significantly. My partnership with Mark Yarvis concerning research on securing distributed adaptation (DARPA-funded) and my involvement with source address validity enforcement (NSF-funded) not only produced publications, but also triggered inspirations.

Frequent interactions with my colleagues at LASR made this research full of wisdom, but also full of fun. Among much appreciation to many, my thanks especially go to Dr. Geoff Kuenning for his help in presenting this research, Dr. Mark Yarvis for sharing his thoughts on writing a good dissertation, Scott Michel for his willingness to test Revere,

Arnell Pablo for setting up testbed machines, Greg Prier, Matt Schnaider and Max Robinson for reviewing a paper on this research, and Jelena Mirkovic for planning a great surprise party after my oral defense.

Janice Wheeler greatly improved the clarity and elegance of this dissertation. I am indebted to her—both for her expert editing work and for always being there when needed.

VITA

1970	Born, Changzhi, Shanxi, China
1988	Graduated Changzhi No. 1 High School, Shanxi, China
1992	B.S. in Computer Science, Peking University, China
1992-1995	Research assistant, Chinese Academy of Sciences
1993-1994	Beijing Duosi Inc.
1995	M.E. in Computer Software, Chinese Academy of Sciences
1995	Awarded Dean's Fellowship, Chinese Academy of Sciences
1995-1996	Researcher, Institute of Software, Chinese Academy of Sciences
1996-2002	Graduate student researcher, Laboratory for Advanced Systems Research, Computer Science Department, University of California, Los Angeles
1997	ACM student member
1998	USENIX student member, Internet Society member
1998	M.S. in Computer Science, University of California, Los Angeles
1999	Summer intern, Network Associates Inc.
2000-2001	Program committee member of ACM SIGSAC New Security Paradigm Workshop
2002	IEEE student member
2002	Outstanding Doctor of Philosophy Degree Award, UCLA Henry Samueli School of Engineering and Applied Science, Computer Science Department
2002-	Assistant Professor, University of Oregon

PUBLICATIONS AND PRESENTATIONS

Jun Li, Jelena Mirkovic, Mengqiu Wang, Peter Reiher, and Lixia Zhang. "SAVE: Source Address Validity Enforcement Protocol," *IEEE Infocom*, 2002.

Jun Li, Peter Reiher, Gerald Popek, Mark Yarvis, and Geoffrey Kuenning. "Position Statement: An Approach to Measuring Large-Scale Distributed Systems," presented at the *IFIP 14th International Conference on Testing of Communicating Systems (TestCom 2002)*, Berlin, Germany, March 2002.

Jun Li, Mark Yarvis, and Peter Reiher. "Securing Distributed Adaptation," *Computer Networks*, S*pecial Issue on Programmable Networks*, Vol. 38, No.3, Elsevier Science, January 2002. pp. 347-371.

Jun Li, Mark Yarvis, and Peter Reiher. "Securing Distributed Adaptation," *Proceedings of the Fourth IEEE Conference on Open Architectures and Network Programming (OPENARCH 2001)*, Anchorage, Alaska, April 2001.

Jun Li and Gerhard Eschelbeck. "Multi-Tier Intrusion Detection System," *UCLA Technical Report CSD-TR-010027*, 2001.

Jun Li, Jelena Mirkovic, Mengqiu Wang, Peter Reiher, and Lixia Zhang. "SAVE: Source Address Validity Enforcement Protocol," *UCLA Technical Report CSD-TR-010004*, 2001.

Jun Li, Xiaoyan Hong, Mani Srivastava, Peter Reiher, and Mario Gerla. "Gradient-Directed Data Dissemination in Wireless Sensor Networks," poster presentation at UCLA Computer Science Department Annual Research Review, April 2000.

Jun Li, Mark Yarvis, and Peter Reiher. "Security in Agent-Based Adaptation," poster presentation at UCLA Computer Science Department Annual Research Review, April 1999.

Jun Li, Peter Reiher, Richard Guy, Gerald Popek, and Geoffrey Kuenning. "Revere—Dissemination of Security Updates," poster presentation at UCLA Computer Science Department Annual Research Review, April 1999.

Jun Li, Mani Srivastava, and Peter Reiher. "Simulation of Gradient-Directed Data Dissemination in Wireless Sensor Networks," *Parsec Workshop*, November 1999.

Jun Li, Peter Reiher, and Gerald Popek. "Securing Information Transmission by Redundancy," *22nd National Information Systems Security Conference*, October 1999.

Jun Li, Peter Reiher, and Gerald Popek. "Securing Information Transmission by Redundancy," *Proceedings of New Security Paradigms Workshop*, ACM SIGSAC, September 1999.

Jun Li and Yufang Sun. "Security Design and Implementation for Micro-Kernel Based File System Server," *Chinese Journal of Computers*, Vol.20, No.5, China Science Press, May 1997. pp. 396-403.

Jun Li and Yufang Sun. "Security Research for Micro-Kernel Based Operating System," *Journal of Software*, Vol.8, No.2, China Science Press, February 1997. pp. 99-106.

Jun Li and Yufang Sun. "Computer Security and Security Model," *Computer Research And Development*, Vol.33, No.4, China Science Press, April 1996. pp. 312-320.

Jun Li. "Security Design and Implementation for COSIX File System Server," master's thesis, June 1995.

Adam Rosenstein, Jun Li, and Siyuan Tong. "MASH: the Multicasting Archie Server Hierarchy," *Computer Communication Review*, Vol.27, No.3, ACM SIGCOMM, July 1997. pp. 5-13.

ABSTRACT OF THE DISSERTATION

Revere—Disseminating Security Updates at Internet Scale

by

Jun Li

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2002

Professor Peter Reiher, Co-chair

Professor Richard Muntz, Co-chair

Rapid and widespread dissemination of security updates throughout the Internet would be invaluable for many purposes, including sending early-warning signals, distributing new virus signatures, updating certificate revocation lists, dispatching event information for intrusion detection systems, etc. However, notifying a large number of machines securely, quickly and with high assurance is very challenging. Such a system must compete with the propagation of threats, handle complexities in large-scale environments, address interruption attacks toward dissemination, and also secure itself.

Revere addresses these problems by building a large-scale, self-organizing resilient overlay network on top of the Internet. This dissertation discusses Revere, and discusses how to secure the dissemination procedure and the overlay network, considering possible attacks and countermeasures. The dissertation presents experimental measurements of a

prototype implementation of Revere gathered using a large-scale-oriented approach. These measurements suggest that Revere can deliver security updates at the required scale, speed and resiliency for a reasonable cost.

CHAPTER 1

Introduction

Over the years the Internet has been seriously challenged by various threats: break-ins, attacks, hoaxes, vulnerabilities, and other malicious subversion efforts. Writers of malicious code, such as viruses, worms, and Trojan horses have been creative in finding ways for their code to propagate rapidly from machine to machine, but defenders of the Internet have been much less aggressive in finding ways to disseminate the information necessary to counter these attacks. As a result, not only have the network infrastructure and individual machines been exposed to various forms of network-based attacks, but they have been slow in reacting to these attacks. This situation raises concerns that were not present when networking was less common and less relied upon.

One critically desirable mechanism for the Internet is to allow a small number of sources, such as trusted centers, to disseminate security information to a vast amount of machines over the network, securely and quickly.

We have developed a system called *Revere* to support such a mechanism. Revere allows a dissemination center to distribute security updates at Internet scale, securely, quickly, and with high assurance.

1.1 Case Studies

1.1.1 An Early-Warning Mechanism

Protection of the information infrastructure is inherently a distributed task. Threats must be countered as a whole, instead of focusing on the protection of every individual

machine. With the rapid growth of networks, a threat gains an increased potential for endangering a larger number of machines, typically through propagation and replication. Each machine connected to network must be aware of all possible attacks. As a result, in a networked environment the threats to an individual machine are of concern to other machines on which the same or similar attacks are also likely to occur.

On the other hand, it is usually the case that a threat or vulnerability that later becomes widespread is first detected in a small number of machines. The difficulty has often been the ineffectiveness of distributing the signatures of or remedies for those threats. Figure 1.1 (excerpted from [CERT 2001:2]) shows that in 12 hours—from 6 a.m. to 6 p.m. on July 19, 2001—over 260,000 machines were quickly corrupted by the CodeRed worm [CERT 2001:1]. An important observation we can draw here is that if

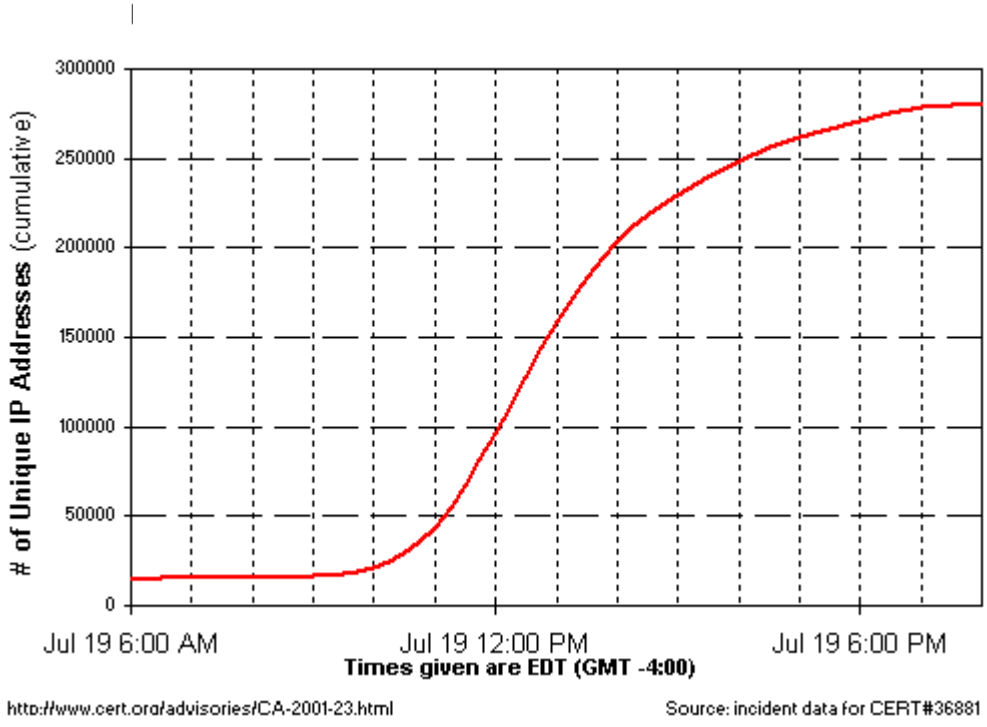


Fig 1.1 IP addresses compromised by the "CodeRed" worm
(data for July 19, 2001 as reported to the CERT/CC)

those machines could have been notified of the incoming attack between 10 and 11 a.m., or even earlier (perhaps between 7 and 8 a.m.), at least 200,000 of them could have been saved. Clearly, the capability to disseminate an early-warning signal to all potential victims of a threat is therefore highly desirable.

The investigation by CAIDA (the Cooperative Association for Internet Data Analysis) after the CodeRed attack was even more astonishing. According to [CAIDA 2001], from July 26 to August 23, 2001, daily examination of a random subset of the 359,000 IP addresses that were originally infected showed that many were still vulnerable to the same attack. Figure 1.2 demonstrates the slowness of those infected machines in patching themselves, and shows that approximately 7% of the machines were still vulnerable at the end of the survey period.

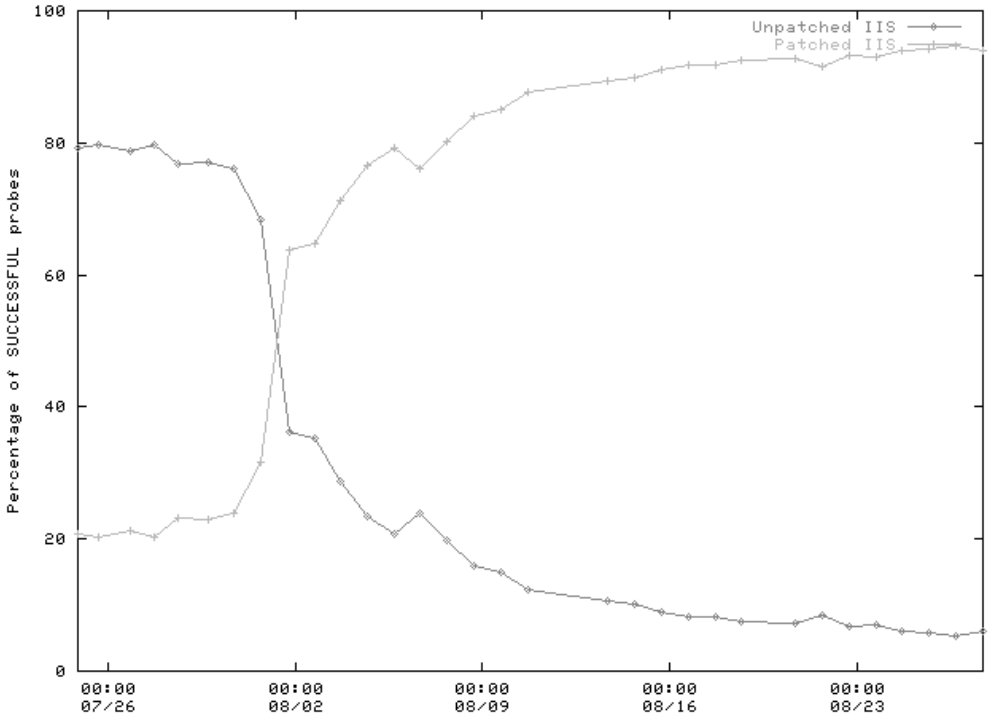


Fig 1.2 Amount of patching in post-infection web servers
(Figure available at http://worm-security-survey.caida.org/)

The CodeRed worm strongly emphasized the need to reliably keep machines up-to-date in order to be resilient to new threats; the CAIDA survey showed the lack of this capability in today's Internet. For instance, during the CodeRed worm incident, `windowsupdate.Microsoft.com` was also infected, and many hosts were reinfected while trying to patch themselves.

Such a capability also echoes the report by the President's Commission on Critical Infrastructure Protection [PCCIP 1997]. After wide investigation and analysis, the commission concluded that the quickest and most effective way to achieve a much higher level of protection from cyber threats is to ensure cooperation and information sharing among the infrastructure owners/operators and appropriate government agencies. In order for this to happen, a real-time attack-warning mechanism must be designed.

1.1.2 Virus Signature Distribution

Computer virus encounters have been increasing steadily over the years. Figure 1.3 shows a computer virus prevalence survey done in the year 2000 by ICSA Labs, a division of ICSA.net. The "love bug," as an example, successfully infected some 3.1 million computer files worldwide by May 5, 2000, according to [BBC News 2000].

Currently, many groups devote substantial efforts to identifying and combating new viruses soon after they are discovered. However, the distribution of information about a newly detected virus is still primitive, and often slow to reach recipients. Typically, a user has to directly contact a virus protection group's web site to download updates, either manually or as scheduled. This pulling-based method often fails to instantaneously keep a user's machine updated, unless the user probes very frequently or the user knows in a timely way that a new virus update has just been published.

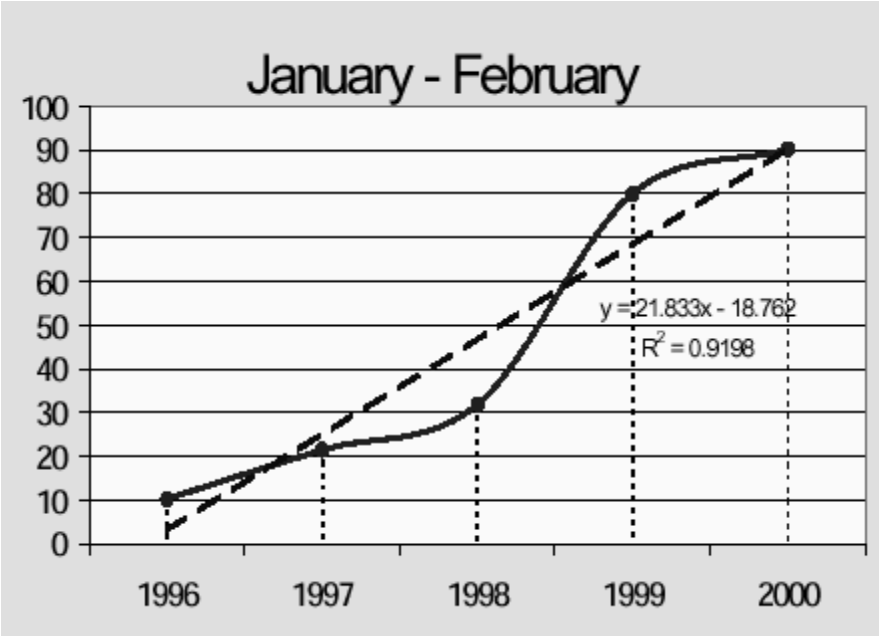


Fig 1.3 Infection per 1,000 computers per month

Given that timely prediction is highly unlikely, “pulling” solutions in high frequency is virtually the only choice here. Unfortunately, this is a suboptimal choice. While attacks in the past have been infecting other machines in the order of minutes, hours, or days, and a user already has to probe a virus center at least as often, one should not rely on an optimistic view that the infection speed of an attack in the future will not be even faster—possibly measured in seconds. As a result, pulling will incur an even higher cost in both CPU time and bandwidth. Such costs can be more prohibitive if a node is interested in receiving hundreds or thousands of different types of security information from various sites.

The process of pulling virus information from a website also lacks flexibility in receiving the information. Because it has to completely rely on the underlying routing protocol which determines the delivery path, a recipient node cannot select its own preferred path for receiving information; nor can it specify more than one delivery path,

both of which can be necessary in some circumstances in order to guarantee the availability of security information.

This pulling method is also not scalable. When millions of users want to simultaneously receive information concerning a new virus, the web site for virus information can become a hot spot of traffic, and users will incur a higher latency due to resource contention.

As an alternative, some groups set up central servers to automatically broadcast new virus signatures to every individual user, but difficulty in managing user records at the central server grew quickly as more users participated. In particular, a new user may want to join at any moment, and an existing user may leave without any notification. Moreover, the center has to send new virus information to every individual user, one by one, a solution that does not use bandwidth wisely.

Recently, peer-to-peer technology has been used to address some of these problems, where a recipient can forward newly received virus information to a second recipient, and a second can forward to a third, and so on [McAfee Rumor]. Every recipient will thus be on a chain of receiving information. However, the design technology to handle disconnected nodes, strengthen security, and maintain the chains has not been reported. Such a system is also subject to man-in-the-middle attacks. For example, if an intermediate node on a virus information delivery path is corrupted, all of its descendents downstream in this peer-to-peer structure will miss the information being delivered.

Clearly, what is needed is a mechanism for rapid and widespread dissemination of new virus signatures, considering important issues such as resiliency, scalability, security, etc.

1.1.3 Information Dispatch in Intrusion Detection

Similar to signature-based virus detection, pattern recognition has been one approach to detecting host-based or network-based intrusions. Newly discovered intrusion patterns also call for a secure and fast dissemination to all those machines that need updates on new types of intrusions.

In addition, a distributed intrusion detection system relies on timely and trustworthy security status updates among individual nodes in order to maintain the state of a system or evaluate a risk level. This again leads to the requirement for a security information dissemination service.

1.1.4 Widespread Certificate Revocation

One difficulty with widespread use of a public key infrastructure has been the certificate revocation problem. If the private key of a machine is compromised, the certificate authority that is responsible for the machine's public key will update the corresponding record and issue a new certificate if a new public key for that machine is generated. However, the stale certificates of the old compromised public key may still be stored in many places across the network. At every node that still uses the obsolete certificate, the compromised public key will be used to verify incoming messages from that machine or protect outgoing messages toward that machine, and both will become vulnerable to attacks.

On January 29-30, 2001, VeriSign erroneously issued two Class 3 code-signing certificates to a person posing as a Microsoft employee [Olavsrud 2001]. Both certificates were assigned to "Microsoft Corporation" and could endorse executable content using keys that claimed to belong to Microsoft. Recognizing the danger that somebody, by impersonating Microsoft, could easily convince users worldwide to

execute an arbitrary program, revoking these two certificates was urgent. Verisign did revoke the two certificates and published them in its current Certificate Revocation List (CRL), but VeriSign's code-signing certificates did not specify a CRL Distribution Point (CDP). Therefore, a user would not know to contact CDP to receive an up-to-date CRL. Microsoft also provided (or suggested) a suite of options, but all of them were not sound solutions to the problem, as explained briefly in the following:

- Asking users to discard certificates dated on January 29 or 30, when asked to confirm the installation of a program. By this option, every user must now read carefully before clicking the "OK" button. And luckily, there were no legal certificates issued on these two dates.
- Developing patches for various Windows platforms. The problem of quickly applying those patches still needs to be solved.
- Removing Verisign from the Trusted Root Store. This would disable any certificates signed by Verisign, and would be a fairly drastic step.

On the other hand, if there were such a service to proactively distribute lists of revoked certificates, every node that subscribes to such a service could then avoid using obsolete certificates. When notified, the node could easily invalidate those cached certificates that should be revoked.

1.1.5 Summary

All of these uses of security information dissemination share many common characteristics, so it is highly preferable to support them with a single common service. These security updates are usually of low volume, but of critical importance. Revere provides this shared service.

1.2 Goal of Revere

The goal of Revere is to disseminate security updates quickly at Internet scale, with high resiliency and robust security. The security updates can contain an early-warning signal, a new virus signature or its remedy, special events in a distributed intrusion detection system, offending characteristics to be filtered by a firewall, certificate revocation lists, and so on.

1.3 Assumptions

This research is based on the following assumptions, and we believe they are reasonable:

- **Revere is only responsible for disseminating security updates.**

  There are separate procedures that generate security updates and independent applications that use security updates. We believe both generation and utilization of security updates are application-specific, and Revere only provides a general service for different applications by addressing their common need.

- **Security updates are usually of small size, at low frequency, and of critical importance.**

  This is true for early-warning signals, virus signatures and remedies, intrusion detection information, certificate revocation lists, etc. One important implication of these characteristics is that the bandwidth cost for disseminating security updates is not a serious concern; one probably can afford to spend several times the minimum required bandwidth if necessary to ensure that the updates are delivered.

- **Revere will run at large scale over heterogeneous nodes.**

  Nodes may have different capabilities and preferences in receiving and forwarding security updates. Also, in a large-scale environment, a significant number of nodes may be disconnected and are therefore not able to listen to security update dissemination for a period of time.

- **Any Revere node could be corrupted.**

  While most Revere nodes will operate correctly, the trustworthiness of any Revere node cannot be assumed without verification. Also, while a dissemination center will normally be well-protected, it could be corrupted as well. However, we do not assume there is a uniform security scheme that can be used by all Revere nodes.

- **Not all Internet nodes run Revere.**

  Any Internet application, including Revere, will not be fully deployed at once. Incremental deployment will be the norm.

- **There is no mandatory requirement for the underlying platforms to run Revere.**

  No changes will be made to operating system, Internet infrastructure, or hardware, in order to support the running of Revere.

1.4 Challenges

Is it feasible to deliver a modest amount of security-related information to most of the connected nodes of an Internet-scale computer network very rapidly, reliably, and securely? Can it be done without huge, powerful server systems? How rapidly can it be done? Within seconds, for example? Does it require fundamental changes to the Internet core or to all end systems, or can it be run purely at the application level?

Revere, or any other system that attempts to deliver rapid security updates at high scale, must overcome several difficult challenges.

Security updates must be delivered at the same speed as attacks, or even faster, if possible. If a node does not receive the most recent security updates, it is highly vulnerable to various threats. On the other hand, if necessary security information (such as worm signatures) were propagated faster than the attack (worm) itself, the threat would be substantially diminished.

Another challenge is scalability. There are tens of millions of machines connected to the Internet, and each machine is a potential participant. Because the scale of the Internet is growing ever larger, a centralized solution would require a single machine, or even dozens of machines, to store global knowledge concerning all potential participants. Even if this were feasible by using powerful machines, the task of keeping the stored information up to date is daunting. Approaches based on centralized management are thus difficult, if not impossible. Distributed solutions can scale well, but bring their own challenges. Further, high scale ensures that significant numbers of nodes will be disconnected at the moment a security update is being disseminated, so any solution must include features to make updates available to those nodes that missed them during dissemination.

High assurance of dissemination is also challenging, especially if a distributed solution is used. Nodes assisting in dissemination may be compromised, resulting in dropped, misdirected or damaged security updates. Approaches such as encryption, authentication, and digital signatures do not actually help ensure that a message is delivered. Attacks that try to destroy or intercept security messages in the middle require other countermeasures. Authenticated acknowledgements are helpful, but do not scale well, and typically retransmitted messages are still subject to interruption threats.

Last but not least, the system itself must be secure. A system that delivers trusted information to millions of machines would become a highly tempting target for attackers. If the system's security is broken, the machines that were targeted for protection will lose that protection. Worse, if the system is widely deployed, it may be used by hostile forces to corrupt even larger numbers of machines.

1.5 Revere Overview

Revere builds an overlay network on top of the Internet for a dissemination center to disseminate any type of security update. This overlay approach provides flexibility, while requiring no changes to existing network infrastructure. Revere is currently implemented as a Java application on participating nodes. These nodes are organized into an overlay network to deliver security updates, where each individual node is allowed to join and leave a Revere overlay network. Every node will receive security updates and every non-leaf node will also forward security updates.

The Revere overlay network is designed to handle an Internet-scale number of participants. First, instead of keeping information concerning all participants of Revere, every node only keeps a small amount of information, typically only related to nodes in its neighborhood on the overlay network and the dissemination center. Second, because high scale means that a significant number of nodes will be disconnected at the moment a security update is disseminated, Revere is also designed to make security updates available to those nodes that missed them during dissemination.

Equally important, the Revere overlay network is resilient. Although it is rooted at a dissemination source, the Revere overlay network is not a tree-like structure. To combat attempts to interrupt dissemination, Revere employs redundancy in the overlay. Whereas redundancy has been widely used in areas like high-availability data storage and some

fault-tolerant systems to provide resiliency, it has been less used to provide network resiliency, in part because of the extra costs of delivering multiple copies of a message. For Revere, the approach is sensible, since Revere is designed to handle a low volume of relatively small but highly important messages. Revere achieves redundancy in the overlay by automatically building multiple as-disjoint-as-possible paths for each node to receive security updates. Revere handles disconnected nodes by providing similarly redundant repository sites that can be contacted when disconnected nodes return to the network.

Furthermore, Revere enforces stringent security for both the dissemination procedure and the overlay network. Each security update will be signed by the dissemination center. If needed, the public key of a dissemination center can be revoked. The security required to build the overlay network is provided by a peer-to-peer security scheme negotiation protocol and a mechanism for pluggable security boxes at each node. These features allow two Revere nodes that do not know each other to communicate without predefining a uniform security scheme.

1.6 Key Contributions of This Research

The first contribution of this research is that it demonstrates that quick and secure dissemination of security updates at Internet scale is feasible.

The second contribution of this research is that of building overlay networks for its special purpose—security update dissemination. Building an overlay network, by itself, is not new. As described later in Chapter 7, Related Work, many researchers have proposed overlay networks for various purposes. However, the special requirements and challenges of disseminating security updates make it hard to simply use any existing overlay networks. Therefore, Revere builds its own overlay networks. Although Revere

allows a node to join or leave a Revere overlay network at its own discretion, just as do many other overlay networks, a Revere overlay network is built and maintained differently. One significant feature is that it implements redundancy in the overlay network in order to address the interruption threats. Every Revere node, at its own discretion, can choose to have more than one security update delivery path that meets certain requirements for speed, security and resiliency.

The third contribution of Revere is that it provides a dual mechanism for security update dissemination: *push* and *pull*. Revere allows a dissemination center to push a new security update to all Revere nodes currently connected, and Revere also allows each individual Revere node to contact repository servers to pull missed security updates. Most information distribution services in a network implement either push operations or pull operations. Revere recognizes that push and pull are complementary to each other, and both must be supported.

The fourth contribution of Revere is that it addresses various possible attacks on the dissemination procedure or the Revere overlay network. Without strong protection, Revere will not be used seriously for receiving important security information. In protecting the dissemination, not only is every security update signed by a dissemination center, thus protecting the integrity of the security update, but also the public key of the center can be revoked in case the public key is identified as corrupted. As for protecting the formation and maintenance of a Revere overlay network, Revere does not assume that a uniform security scheme will be enforced across all participants; instead, a peer-to-peer security scheme negotiation protocol is designed. Every node can specify what security schemes to follow for its incoming messages. Furthermore, a Revere node can plug in a security box for every particular security scheme, making it easy to enforce many different security schemes.

The fifth contribution of Revere lies in the technique for measurement, something that can also be applied to other distributed applications. Because of the intended scale of the system, direct measurement of Revere is impossible at this point. Therefore, Revere is measured by using an "overloading" technique. With this technique, a physical machine can host many nodes of a distributed system; here, each logical node still runs the real code, just as it would in the real world, except that every logical node sits on top of a *virtual topology*. Large scale can then be achieved using multiple physical machines, each supporting many logical nodes.

The sixth contribution of Revere is that it provides an easily deployable solution. Implemented in Java at application layer, Revere does not require any changes to the network infrastructure underneath, and does not require any particular support from the OS or hardware to be deployed. Any user, who has installed Revere on its machine, can just begin running Revere in order to join a Revere overlay network, or can withdraw from the system by clicking a button.

1.7 Roadmap of This Dissertation

We describe and discuss Revere in more detail in the following chapters. Chapter 2 discusses the general principle of securing information transmission by using redundancy. Chapter 3 discusses the *RBone*, the overlay network that Revere builds for security update dissemination, and the dissemination procedure itself is described in Chapter 4.

We identify security issues and discuss our approaches in Chapter 5, where we outline possible attacks and discuss their countermeasures. Both protection of the dissemination procedure and the protection of the overlay network will be addressed. In Chapter 6, we introduce a large-scale-oriented "overloading" approach to measuring

large-scale distributed systems, and apply it to Revere measurement. We will discuss what metrics should be evaluated and what measurement procedure we have taken. We then report and analyze the results of our measurement.

Chapter 7 summarizes related work, including various information distribution approaches viewed in the most general context, practices on virus signature distribution, overlay networks developed by other people, multi-path routing, etc. Chapter 8 is devoted to future work, where we will show that Revere provides a good platform for interesting research along several lines. We conclude the dissertation in Chapter 9.

CHAPTER 2

Assurance via Redundancy

One of the most critical challenges facing Revere is that of supporting the high availability of security update dissemination under various circumstances, including the case where an attacker is trying to corrupt information while in transit. In this chapter we justify the fundamental concept of information assurance via redundancy and discuss general considerations on using redundancy to secure transmissions; we will thus establish that a redundancy mechanism is critical to the success of Revere.

2.1 The Redundancy Approach

More and more information is now shared and distributed over computer networks. Secure distribution of such information is becoming increasingly important. Conventional security approaches address many of the problems of securing information dissemination, but not all of them.

Encryption can provide secrecy, authentication can provide assurance of the source, digital signatures can provide integrity verification, firewalls can filter out dangerous transmissions, and so on. But these and other traditional mechanisms offer little assistance with interruption threats. No matter how elaborate the encryption or authentication, if the information is dropped on the floor, destroyed or transformed into a piece of garbage, blocked due to overloading of an intermediate link, or disrupted by other malicious acts, information availability is damaged. In many cases, even if

attackers cannot decrypt, forge, or alter information, they can achieve their ends merely by ensuring that important information does not reach an intended destination.

The traditional solution is to require acknowledgement of important messages. Since attackers might try to forge acknowledgements, they are typically signed (and possibly encrypted, if they contain sensitive information). If an acknowledgement is not received soon enough, the message is resent. This method works well if a relatively small number of messages require acknowledgement. If a very large number of messages must be acknowledged, then hierarchical or other load distribution methods must spread out the responsibility for checking acknowledgements. In the general case, all nodes performing the checks must be trusted.

Worse, an attacker can repeatedly intercept or destroy the retransmitted message. Without other mechanisms, an attacker who has compromised a single link or router node may permanently prevent the delivery of a message, since each retransmission will probably still follow the same path through the compromised resource.

We believe the fundamental problem is that there is only a single path for information transmission. If any point of this single path is corrupted, transmission security is corrupted. This problem can be reduced by adding redundancy to information transmission structures. Such redundancy can improve transmission resiliency and greatly improve information availability. Typically, such redundancy can be provided by using more than one path through the network to reach the destination.

If redundant paths are completely disjoint, then attackers must compromise multiple resources in the network to prevent message delivery. A greater degree of redundancy means that more resources must be compromised by attackers. Assuming that there is cost and risk in compromising each resource, increasing the degree of redundancy can thus increase the difficulty of preventing successful delivery. Obviously, redundancy

uses more resources than single-path transmission, and there is a tradeoff between the degree of security achieved and the cost of providing it.

Similar arguments have demonstrated the value of redundancy for many hardware fault tolerance problems. In the networking realm, however, actually providing true redundancy may be difficult. While two distinct disks can be used for storing the same data, or two distinct processors can be loaded with the same instructions, it is not always true that two or more disjoint paths can be easily found for reaching a specific destination through a network. Such paths might not exist. Even if they do, existing network routing protocols and the desire to hide network complexities from higher levels make discovering and using the disjoint paths difficult. And it is even more difficult to know the locations of those physical lines that a message follows.

On the other hand, an Internet-like network often comes with abundant routes and connections in order to be resilient to faults and failures (in particular, this is the basis for enabling routing protocols to select specific routes to reach a destination). The Internet, for example, is a worldwide mesh or matrix of hundreds of thousands of networks that are interconnected by about 8,000 ISPs (Internet Service Providers) at the core [Quarterman *et al.*]. Even at the edge of the Internet, more and more organizations have become multi-homed, with connections to multiple ISPs.

In a word, we believe that redundancy can have great value in counteracting attacks. Even if the paths are not fully disjoint, any non-shared portions of the path limit an attacker's choice of attack points. The attacker must either find and compromise shared links or routers on the path, or must compromise the right set of non-shared elements. The volatility and obscurity that makes finding disjoint paths difficult also makes attacking them hard. While some choke points cannot be avoided, link-by-link (or segment-by-segment) redundancy may still prove very useful.

Redundancy for fault-tolerant information transmission has been studied by many people [Castro *et al.* 1999] [Pelc 1996]. Dealing with Byzantine faults has also been considered. However, this research only focused on specially structured networks, such as broadcasting over complete networks or hypercube.

Another related research area is information dispersal [Rabin 1989]. This has some similarities to the RAID technology for data storage. The original information is divided into pieces with some level of redundancy before being transmitted separately. After obtaining the pieces, the receiver can assemble them into the original information, even if some pieces are lost or damaged. But, if all these pieces reach the receiver via the same single path, every piece will still be subject to interruption threats, causing the assembling operations to still fail.

2.2 Interruption Threats

While information transmission latency has been shortened dramatically in the past few years, information transmission may still have to cut across several external entities or domains. A malicious attack might be able to penetrate a network element where everything seems under control. These external or maliciously penetrated places are where interruption threats will occur.

Interruption threats can be divided into two categories: path interruption and data interruption. A path interruption happens when information is dropped on the floor or misdirected to the wrong place. A path interruption also happens when some portion of the transmission path is flooded, causing denial of service. A data interruption happens when the information itself is damaged.

A recipient usually has a better chance of detecting data interruption than it does path interruption. A recipient with a data interruption can detect that data has been

manipulated, while a recipient with a path interruption may not notice anything abnormal at all. The commonality is that both types of interruption can happen even if the malicious entity does not know what the data contains, so neither encrypting nor signing can help. Interruption threats can be more serious if combined with other kinds of security attacks.

2.3 Transmission Primitives

Some transmission primitives have addressed the difficulties in transmitting information. While these primitives are designed mainly for non-security reasons (particularly reliability in the sense of no data loss or physical error) and they do not provide a total solution, they do provide some assistance in coping with security problems in information transmission.

In the following subsections, we will discuss TCP, reliable multicast, broadcast and flooding, and show that those efforts which deal with transmission difficulties are insufficient to address interruption threats.

2.3.1 Reliable Transmission – TCP

TCP [Postel 1981] provides reliable one-to-one information transmission on top of the IP layer, where an IP packet is routed to the destination along a dynamically determined physical path. If a TCP packet is lost according to acknowledgement information from the receiver, or if its own retransmission timer times out, a TCP sender retransmits the TCP packet.

If interruption attacks are sporadic, causing TCP to drop an occasional packet or sometimes damage the data, a TCP retransmission can heal the problem. But essentially TCP cannot eliminate interruption threats if the retransmitted TCP packets encapsulated

in IP packets are sent through the same hostile point and are maliciously manipulated again.

Inspecting the TCP acknowledgement mechanism will show that TCP retransmission appears even less effective in addressing interruption threats. In the reverse direction of a TCP connection, the acknowledgement packets could also be subject to interruption threats; for instance, when a TCP connection is symmetric and the reverse routes passes through the same hostile point. An acknowledgement packet, even with a signature or other security enhancement, may not be able to reach the TCP sender smoothly. If so, the sender will not be aware of packet loss, damage or misdirection, and the sender will not retransmit before it times out.

Even if a sender retransmits (either after it times out or after it detects packet loss based on an acknowledgement), a retransmitted packet often uses the same path as the original one. This causes the packet to cross the compromised point on the path again, and thus be subjected to repeated interruption threats.

2.3.2 Reliable Multicast

Reliable multicast provides reliability for information multicast. Usually it is done by negative acknowledgements or repair requests. As one example, SRM (scalable reliable multicast) [Floyd *et al.* 1995] lets each multicast recipient be responsible for information loss or error by requesting a repair from the whole multicast group (not necessarily from the sender) or by initiating a local recovery. Since a multicast group could be flooded when every recipient sends a repair request to the whole group for the same repair, SRM suppresses repeated repair requests and allows only one copy to be propagated throughout the group. Similarly, repairs are also suppressed.

Although this bandwidth-aware reliability mechanism is reasonable in normal conditions, it is susceptible to interruption threats. Designed to address packet loss due to transmission errors, it cannot successfully handle packet loss or damage due to malicious efforts. Although repairing packets can be injected into a multicast group, reliable multicast does not provide alternative paths for packet delivery, and a repair packet can still be interrupted.

2.3.3 Broadcasting and Flooding

Broadcasting and flooding are used to reach multiple destinations with a best effort in a transmission session. A recipient may receive more than one copy of exactly the same information, which inadvertently gives rise to some level of redundancy (perhaps not enough) by heavy use of bandwidth. Standard broadcast and flooding methods assume that all members are benign nodes and that they follow the rules for transmission.

When dealing with the Internet, broadcast and flooding will also introduce other problems. Broadcast is typically done at subnet level, but the number of subnets over the Internet is still large, making broadcast a non-scalable approach. Flooding at Internet scale is also daunting, since this would incur a prohibitive bandwidth cost.

Reliable broadcast [Chang *et al.* 1984] has been proposed to deal with information loss or error caused by non-security problems such as physical transmission errors. Obviously it cannot eliminate interruption threats for the same reason that TCP cannot.

In a word, conventional approaches to information transmission can provide good reliability in terms of “natural” information loss or error, but provide little support in counteracting “artificial” information loss or damage. To deal with these interruption threats, we need a new approach to transmitting information in a secure fashion.

2.4 Assurance via Redundancy

We propose that redundancy in information transmission is valuable for providing security assurance [Li *et al.* 1999]. Here, redundancy means that the information transmission path, or part of the path, is multiplied to avoid a single point of security corruption.

We believe redundancy is important for security assurance in large-scale networks like the Internet. While massive redundancy in a small-scale environment may be employed to achieve best resiliency, lean but resilient redundancy is the fundamental goal for security. Brute-force redundancy will result in an uncontrolled waste of resources in a large-scale environment, which in turn may overload some resources to cause denial of service.

2.4.1 Redundancy in Other Areas

Redundancy has been widely used in many areas by devoting more resources to achieve better availability. Resource redundancy is often applied to include multiple processes, multiple hardware components, and multiple data copies, usually with independent failure probabilities. Examples include:

- *High availability data storage*. Here, in order to deal with disk crashes, balance load from a hotspot disk, and to provide lower latency for data access, either more than one disk stores a copy of the data, or the data, with built-in redundancy, is dispersed to more than one disk. This is normally transparent to users [Patterson *et al.* 1989].

- *File replication.* This process is used to make replicas to support easier access [Kuenning *et al.* 1997] [McDermott 1997] [Reiher *et al.* 1996]. Establishing mirror web sites for lower latency is one such example.

- *Data backup*. Data backup, usually done periodically, can help restore damaged or lost files from backed-up copies.

- *Fault-tolerant distributed systems*. Here, corresponding to a task, multiple replicated executions [Singhal *et al.* 1994] may be employed to run a program concurrently at different locations. The task can still smoothly continue if at least one execution succeeds.

- *Mapping one web site to multiple IP addresses*. Mapping one web site to several different server machines, in a round-robin fashion or in some other more sophisticated way, can prevent one single server from being overloaded and ensure that the site is accessible even if some server machines have crashed [Alteon].

2.4.2 Resiliency Evaluation

Given a graph with fault probability distribution of nodes, computation of the probability that there is a non-faulty path between two arbitrary nodes is known to be NP-hard in the worst case. But we can still look at some resiliency properties of a graph for some basic understanding of which redundancy structures are good.

Let us define the resiliency of a one-to-one connection as the probability that the source *S* can *reach* destination *D*, denoted as $R_{S\text{-}D}$. Here, the word "reach" means that when every path from *S* to *D* is used to transmit a copy of a message at the same time, at least one authentic copy can be received. Further assume for this specific connection that there is a total of *m* cutsets C_1, C_2, C_3… C_m, each containing some number of elements (a single element cutset corresponds to a choke point, for instance). Denote E_i (i=1, …, m) as the event where at least one element of C_i is not broken, then we can define:

$$R_{S\text{-}D} = \textit{Probability}\ (E_1 \text{ and } E_2 \ldots \text{and } E_m)$$

Usually decreasing the number of cutsets, here *m*, can increase the resiliency of a connection. Further analysis can also show that higher resiliency can result if a cutset contains more elements, or an element has a lower probability of being subverted.

Having each path be as strong as possible by passing through the least number of corrupted nodes can decrease the number of cutsets; and having more paths, in particular as disjoint as possible, to a destination can make a cutset of the connection contain more elements, thus strengthening the resiliency of the connection in general.

2.4.3 Using Redundancy in Transmission

Redundancy may be improved by simply increasing the number of information sources or the number of transmission paths, particularly when information corruption is detected.

This may not provide extra security assurance in information transmission, however, and may lead to unwise resource usage and degraded performance. For instance, if the incoming link for a receiver is maliciously flooded, causing denial of service, contacting more sites for redundant information may not yield any useful message; it may instead cause even more severe overloading of the link.

Therefore, to achieve the best information delivery assurance, a sophisticated redundancy design is necessary. The designer should understand the stochastic distribution of interruption threats, make the best trade-off between resource usage and redundancy, build resource-saving but resilient transmission structures, use an adaptive algorithm to help choose when and how to deploy redundancy, and so on. For instance, a receiver may also run an intrusion detection facility to find the reason for continuous information unavailability.

There are many complex issues in deploying redundancy in large-scale networks like the Internet. One problem is that machines in the Internet are heterogeneous in terms of transmission characteristics, platform, security situations and requirements. Ideally, some of this information should be taken into account when choosing redundant paths. For example, if a particular node is suspected of being highly insecure, special care should be taken to avoid routing multiple supposedly disjoint paths through that node. Also, the security system must be adaptive in dealing with a dynamic environment in terms of location, transmission mechanism, and impact of interruption threats.

Further complexity arises because a compromised element can compromise other intermediate elements or cause them to misbehave. For instance, while misbehaving on data traffic itself, a compromised router may cause other routers to unknowingly misbehave by sending them false routing messages [Wang *et al.* 1997]. Building security into the routing infrastructure is itself a challenging task [Cheung *et al.* 1997] [Jou *et al.* 1997] [Wu *et al.* 1998] [S-BGP]. Unless the routing infrastructure security is strong, two paths used to reach a destination should not only be as disjoint as possible, but also isolated within the routing infrastructure. For instance, using routers belonging to different ISPs would be preferable.

Last, as we pointed out earlier, resilient but lean redundancy is what we want. Obviously, in a large-scale network such as the Internet, building such a structure can only be done in a distributed fashion, adding further difficulty.

2.5 Employing Redundancy in Revere

Revere employs redundancy in several aspects: (1) every Revere node can have multiple security update delivery paths; (2) every Revere node can contact several sources for missed security updates.

In the first case, a Revere node will try to guarantee that one path will provide the fastest delivery of security updates, and the other paths will provide the best resiliency. We will address this in more detail when we discuss the Revere overlay network formation and management.

In the second case, having several independent sources for old security updates supports higher availability. If one source is subverted, another source may still be able to provide complete authentic security updates. We will describe this more when we discuss the dissemination procedure. Both cases will also be revisited when we discuss possible attacks toward Revere and countermeasures.

More interestingly, Revere runs at Internet scale. Since a centralized solution to building a good redundant distribution structure is not feasible, a distributed algorithm that builds a Revere overlay network on the fly must be designed. The security of this distributed algorithm is indispensable. If the redundancy mechanism is compromised, the supposedly beneficial system could actually work against security. Some problems in this area and solutions to the problems are obvious, but more subtle and indirect problems are likely to occur. Theoretical understanding of large-scale redundancy for Revere also requires investigation. For example, in addition to evaluating the resiliency of a one-to-one connection, how does one evaluate the overall resiliency of a dissemination structure?

Revere does not address hardware-level redundancy. When a Revere node obtains multiple delivery paths for receiving security updates, those paths might overlap at a hardware level, even though they are quite disjoint at a logical level. Some research has shown that in general, logical-level disjointedness matches nicely to hardware-level disjointedness [Andersen *et al.* 2001]. Nevertheless, due to the difficulty in gaining

hardware-level topology information and other relevant knowledge, Revere leaves this as an open issue. We will investigate this more in future work.

2.6 Conclusions

In this chapter we discussed using redundancy to secure information transmission against interruption threats. Our analysis shows that both conventional transmission primitives and frequently used security techniques are not adequate when counteracting interruption threats. Redundancy, an approach already widely used in many areas but rarely in information transmission, can actually improve the security of information transmission. While redundancy has a wide applicability in many areas of network security, we have outlined how redundancy can also be used in Revere for security update dissemination.

CHAPTER 3

RBone: A Self-Organized Resilient Overlay Network

An RBone must be formed and maintained for each different type of security update notification to ensure delivery of the needed security updates. Composed of Revere nodes and the logical links between them, an RBone is the basis of security update dissemination. During security update dissemination, Revere forwards security updates from one node on the RBone to another along the virtual link between them.

An RBone organizes itself. By merely using a simple user interface, Revere allows each individual node to join or leave an RBone automatically. Revere allows a node to attach itself as a child of other existing Revere nodes to become part of an RBone. Revere can also detect broken or dead nodes on an RBone and handle broken links on the RBone, causing nodes to reattach themselves as required.

Because Revere works at the application level, the RBone built by Revere achieves resiliency at the same level, considering only other participating Revere nodes. Since information about general network topology is usually unavailable, Revere does not attempt to achieve hardware-level disjointness of paths. Mechanisms to achieve hardware-level disjointness are a topic of future research.

To achieve both efficiency and resiliency, Revere allows an individual node to select more than one parent, with one of parents providing the fastest security update delivery and the rest delivering copies along paths as disjoint as possible. Thus, a node will only miss security updates if all its paths are broken.

Given that an RBone can contain millions of nodes, all RBone management operations must be as simple as possible and rely only on a small amount of partial knowledge at each node to support scalability. In Revere, each node only keeps information about its parents, its children, and the dissemination center. Each node is able to choose its number of parents and children.

For convenience, we assume that a different RBone rooted at a specific dissemination center will be built for every different type of security update. Sharing a common RBone for different types of security updates and different centers leads to some complexities, which we address briefly at the end of this chapter.

In the following, we discuss the principles of organizing the nodes on an RBone, the formation procedure of an RBone, and the mechanism of maintaining an RBone. We will also describe the message format and data structures used in RBone formation and maintenance by each Revere node. We leave the discussion on the security of RBone to Chapter 5.

3.1 Principles for Building an RBone

While ensuring that an RBone must be secure, fast, scalable, and lightweight in terms of its characteristics, there are three prominent principles to follow in terms of building a basic RBone. First, an RBone is an overlay network. Second, an RBone is self-organized. Third, an RBone must be resilient.

3.1.1 An Overlay Network

Revere faces a list of challenges. As discussed in Chapter 1, Revere must be fast, resilient, scalable, and secure. Yet there are no known mechanisms, including those related works to be investigated in Chapter 7, to address all these challenges. For example, any pulling-based mechanism, which requires each node to pull security

updates from a dissemination center, will fail to instantaneously keep nodes updated unless every node probes the center very frequently or knows the availability of new security updates immediately. Any unicast-based dissemination mechanism, if requiring the dissemination center to directly send security updates to each individual node, forces the center to record the identities of all nodes that want to be updated, violating the scalability requirement. IP multicast, whose group-based subscription paradigm frees a dissemination center from recording the individual identities of any joined nodes, is however, based on a tree-like structure for disseminating information, and has little resiliency and is susceptible to interruption threats. The deployment of IP multicast has also been very difficult [Francis 2000]. Broadcasting mechanisms, some of them designed to be resilient, are typically only applicable to local area networks and not portable to the whole Internet. Worse, security issues arise in all these candidate mechanisms and must be addressed.

We believe that in order to address all these challenges, a special network must be built. Physically wiring such a network is infeasible. Logically, an alternative is to build into routers some special mechanism that can provide Revere-like service, such as multi-path forwarding of security updates. However, this results in several problems. First, conceptually this violates the layering model of the Internet; routers should be used for general-purpose packet-forwarding service at the IP level, not for special-purpose services like Revere. Every node that needs security updates typically has its own specific requirements or preferences regarding receiving security updates. Having routers address those particularities may cause very complex operations. Second, this would face a deployment challenge; adding new software to routers is known to be a difficult problem. A close example is IP multicast, which is still striving to achieve widespread router-level implementation after years of efforts. Third, such a router-level

network for security update dissemination, if implemented, would require a significant effort to manage and secure. This makes Revere face a dilemma: if Revere-enabled routers do not enforce security well, Revere, as a high fan-out dissemination system, can propagate corrupted or replayed security updates to a large number of security update recipients; but if Revere-enabled routers enforce security, security issues such as key management, node authentication, message verification, and replay prevention must be handled by those Revere-enabled routers.

We propose to build an overlay network on top of the Internet, corresponding to each special type of security update (Figure 3.1). Such an overlay network will be composed of all nodes that "subscribe" to receive those security updates. On top of the Internet, each node will run Revere at application level, which greatly enhances the flexibility of each Revere node in terms of configuration and adding new functionalities. An application-level service will also be easy to deploy: a node can simply install Revere and start running it in the same way as normal applications. An application-level service will be easy to debug as well. More important, we can now start designing this overlay network to meet all the challenges that Revere faces.

Every Revere node on this overlay network could just be a leaf node that purely

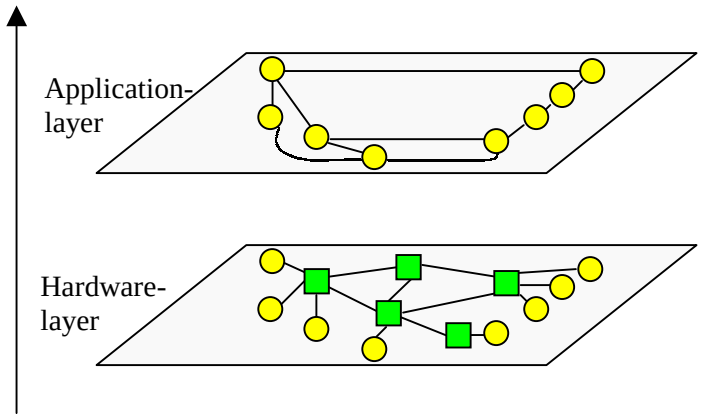


Fig 3.1 Application-layer overlay network
on top of hardware layer

listens to new security updates. Unfortunately, this will turn an RBone effectively into a unicast-based star-like structure, with a dissemination center as the source for every node. Such a structure will require a dissemination center to store the addresses of all Revere nodes and transmit security updates to each individual node, one by one, and will lack redundancy as we discussed earlier. As a result, it is obviously not scalable, not efficient, and does not support transmission redundancy.

Instead, Revere allows each node to forward security updates to others, instead of relying only on a dissemination center. Therefore, an RBone will be composed of some middle Revere nodes "inside," which forward security updates to other nodes, and some leaf Revere nodes "outside," which simply receive updates. As we will illustrate later, each node can choose to have multiple RBone nodes as its parents, corresponding to multiple as-disjoint-as-possible dissemination paths.

Scalability is also much easier to address now. By allowing each node on an RBone to forward security updates to other nodes, security updates can be propagated hop by hop. As a result, a dissemination center, or any other node, does not have to remember the identities of all security update recipients, save to remember the keys of all other nodes in order to verify the authenticity of all nodes. This hop-by-hop transmission characteristic allows each node only to record its parents and children on an RBone.

3.1.2 A Self-Organized Network

Consisting of Revere nodes and directed links between them, an RBone is dynamic. New nodes may wish to join the RBone at any time. On the other hand, an existing node may crash, become corrupted, or simply want to leave the RBone. Some existing nodes may also be disconnected for some time, and then come back. Revere needs to adapt itself to all these situations quickly through a distributed cooperation.

The answer to this problem is to let the RBone organize itself. When a new node wants to join Revere, it should identify and then contact some existing Revere nodes on an RBone, and then try to attach itself to the RBone at the best possible position, finally becoming a child of one or more existing Revere nodes. When an existing node quits, disconnects, crashes, or misbehaves, other nodes associated with it should detect the anomaly and isolate themselves from this node. The isolating actions allow each parent of this broken node to dismiss the node as a child, and each child of this broken node to attach to a new parent. Similarly, when a child node detects that one of its parents becomes too distant, the node may also choose to detach itself from this parent and attach to a closer one, in order to achieve better dissemination efficiency.

With such self-organization, no manual intervention is required to manage an RBone. Some problems remain to be solved, however. For example, how can a node decide whether to attach itself to another node? Or, since an RBone can comprise millions of nodes, for scalability consideration, the self-organization of an RBone must rely only on partial knowledge at each node; thus, how can an RBone self-organize itself based on partial knowledge?

3.1.3 A Resilient Network

Revere is designed to disseminate security updates in a hostile environment. Because of the importance of security updates, Revere nodes desire high assurance relative to receiving security updates, even if the data may have to come through a hostile environment with all kinds of possible attacks.

There are many approaches to providing strong resiliency. One possible approach to building a resilient network is to use some feedback mechanism, such as signed acknowledgements or signed negative acknowledgements. However, this is difficult.

First, being a general problem as discussed in Chapter 2, single-path dissemination or feedback is subject to interruption threats. Second, the large scale with the vast quantity of participants makes verification of positive or negative acknowledgements from these participants even harder. The originator of the update, or any verifying server, probably cannot handle millions of acknowledgements, especially when doing so requires cryptographic authentication. Even with a distributed or hierarchical method, if a verifying server had to maintain all necessary keys, a heavy overhead would result. Additionally, the protection of verification servers adds new problems. Third, due to the lack of trust, a feedback-based approach would require that all participants have trusted keys, leading to a huge key distribution and management problem. Last, as for using negative acknowledgements to avoid an acknowledgement explosion, unfortunately a recipient won't send a negative acknowledgement at all if it does not know that it *should* have received an update. This approach only works when it is feasible for a receiver to realize that he has not received a security update that was sent. Security updates, for the most part, will not be sent periodically, but on the occurrence of unusual or unpredictable events. Thus, Revere nodes will have no hint of when they might want to generate a negative acknowledgment. Also, a negative acknowledgment has the same problem as a positive acknowledgement in preventing itself from getting dropped.

Instead, as we discussed in Chapter 2, we propose to build redundancy into the RBone to achieve resiliency. Each node interested in receiving security updates can choose to attach itself to an RBone and obtain multiple copies of security updates to achieve security assurance (Figure 3.2 shows a mini RBone where some nodes have more than one paths). Thus, if a node can receive data from multiple disjoint paths, only when *all* paths are corrupted will the receiver be prevented from receiving the data.

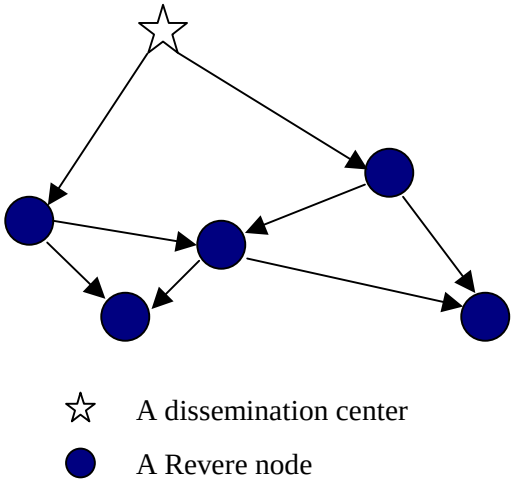


Fig 3.2 An RBone rooted at a dissemination center

While it seems that this method wastes bandwidth, the disseminated security updates are usually of small size, low frequency but vital importance. As long as the number of disjoint paths is not too high, using redundancy is acceptable. Later in this chapter we will describe how to compose or detect disjoint paths.

3.2 RBone Formation

Starting from only a dissemination center, an RBone is formed as more and more nodes join. During its join procedure, a new Revere node has to locate some existing Revere nodes first, negotiate with them, and decide which of those nodes together, as parents, can provide the best efficiency and resiliency. In Section 3.2.1 below, we first address the search for existing Revere nodes, and in Section 3.2.2, we describe the three-way-handshake negotiation procedure, the basic operation during a node's join procedure. In Section 3.2.3, we discuss the life cycle of a join procedure, especially when a join procedure ends. Parent selection, a key component of the three-way-handshake, will be further illustrated in Section 3.2.4.

3.2.1 Finding Existing Revere Nodes

To join an RBone, it is necessary for a new node to contact some nodes that have already joined the RBone. Revere allows each new node to use its own method to search for those nodes. Various methods can be employed, such as using configured knowledge (for example, the address of the dissemination center or a local designated Revere node), contacting a directory service, applying a multicast-based expanding-ring or expanding-wheel search [Rosenstein *et al.* 1997], or adopting lookup services used in peer-to-peer systems [Ratnasamy *et al.* 2001] [Stoica *et al.* 2001]. Here we describe and analyze several exemplary methods of node searching:

- **Configured knowledge**

  A new node can start up with the knowledge of some existing Revere nodes. For example, the dissemination center that provides security updates to every node is publicly known already. The new node can simply contact this node. Or, in the local administrative domain of the new node, there may be a node that is already on the RBone and configured as the initial contact for new nodes wanting to join Revere.

- **Directory service**

  A directory server can be established to store information regarding some existing Revere nodes. A new node, in order to locate some existing Revere nodes, can send a query to this directory server. Included in the query could be the node's IP address, the type of security updates that it needs, the number of existing nodes that it requires, the method for node-to-node communication that it prefers, the security scheme that it enforces, etc. Upon the receipt of a query, the directory server can then check its

database of existing Revere nodes; once matches are found, the server then sends a response back to the new node with a list of existing Revere nodes.

Given that Revere is a large-scale service extending over the entire Internet, a single directory server faces a serious scalability problem. First, the directory server may be overloaded by handling an overwhelming number of requests. The directory service may be replicated, but this leads to database synchronization issues between different replicas. Second, maintaining the database of a directory server is not simple. After a new node joins an RBone, its parents, or the new node itself, or some third party, should notify the directory server of the new membership. Or, after an existing node quits an RBone, a withdrawal notice should be sent to the directory server as well. Worse, some nodes may become disconnected or broken without being able to notify the directory server, resulting in obsolete information at the directory server.

To provide scalable directory service, another alternative is to build a distributed directory service. For example, each administrative domain can have a directory server, responsible for maintaining the information of existing Revere nodes inside the domain and handling requests also from inside the domain. This, however, limits the scope within which a new Revere node can search for existing Revere nodes, and adds a management burden to Revere.

- **Multicast-based expanding-disc search**[1]

  All of the above problems have led to various IP-multicast-based methods. IP multicast, first proposed in [Deering 1989], is a mechanism by which packets can be efficiently routed from one source to many destinations. Other than the efficiency with which queries may be distributed to many destinations simultaneously, there are two other advantages afforded by multicast for this application:

  - A client may query the set of servers without knowing their explicit locations (this capability is available irrespective of the IP multicast routing protocol).
  - A client may use TTL-based scope control in order to contact the topographically closest servers first.

  Given the capabilities of multicasting, it is possible to design a system that efficiently transports directory queries to a set of distributed directory servers. These servers would ideally be located at the actual sources of the directory information (e.g., one directory server at each Revere node site). There is at least one such implementation, called *march* [Kashima 1995]. *march* is a multicasting distributed directory database system that relies on a single multicast address for all directory servers. It uses TTL scope-limiting to constrain the impact of individual queries. By iteratively expanding the TTL, a *march* client finds the closest (topographically speaking) FTP site containing the requested information. This type of

[1] Discussions on multicast-based expanding-disc search and multicast-based expanding-wheel search are based on "MASH: the Multicasting Archie Server Hierarchy" by Adam Rosenstein, Jun Li, and Siyuan Tong, which appeared in *Computer Communication Review*, Vol. 27, No. 3, ACM SIGCOMM, July 1997.

search is often referred to as an “expanding-ring” search. Expanding-ring searches are inherently robust, as any servers that fail to receive a query during an iteration have another opportunity to receive the query during the next iteration.

However, there are drawbacks to the *march* approach. In each iteration of an expanding-ring search, queries must be routed to all *march* servers that were reached on previous iterations. Such flooding (even inside a limited TTL radius) of a pervasive multicast group can result in unnecessary traffic. Owing to this re-querying of the inner (previously queried) search rings, such searches may be more aptly referred to as “expanding-disc” searches. Although the inner disc does not service repeated queries in the *march* architecture, the routers do not understand this and must still route all repeated queries to the same servers at potentially great cost with no additional benefit. Should *march* become a popular service, this poor scaling factor could contribute to Internet congestion.

- **Multicast-based expanding-wheel search—*MASH* approach**

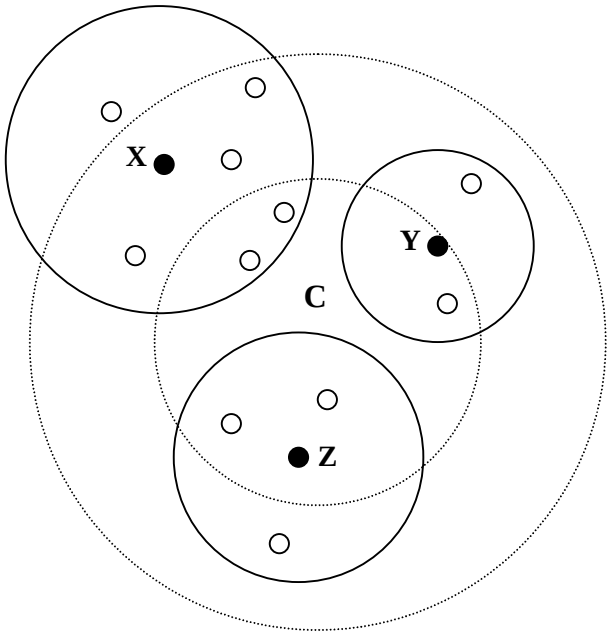
  [Rosenstein *et al.*1997] developed a *MASH* service to search for FTP sites that contains a particular file. This MASH approach can be generalized for the purpose of searching existing Revere nodes. By constructing a two-level hierarchy of *march* servers, MASH employs a hierarchical approach to address the problems posed by expanding-disc searches. This service is characterized by one well-known, pervasive multicast group G_{global}, and a number of topographically localized subgroups. The well-known group is much like *march*'s group, but its number of members is

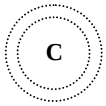
comparatively small, and is self-adjusting as the number of servers in operation scales up. The subgroups each have their own multicast address. The servers dynamically organize themselves into these groups. Each group includes one (only) member of the pervasive group. This "parent member" receives queries from clients on the global multicast address, G_{global}, and dispatches these queries to its subordinate servers via its unique local group multicast address.

In expanding-disc searches, the client completely controls the impact of its searches. Since only a TTL limit is used, maximum radius is the only dimension that may limit the scope of a search. In a hierarchy, each hierarchical layer can share the burden of restricting the search scope. What results is a search pattern whose impact, with respect to the multicast traffic it generates, is greatest at its frontier, and restricted to minimal "spokes" en route to the frontier. Thus, this search is called an "expanding-wheel" search.

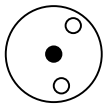
Figure 3.3 shows an example of this approach. First, the client C sets its TTL limit to some small initial value (the dashed inner circle) and transmits its query to the global multicast address. Only the root level servers (dark filled-in circles) listen to this address.

In Figure 3.3, C's first transmission will reach servers Y and Z. These servers will either respond themselves (if their local databases match the query) or they will retransmit C's query by multicasting it to their respective groups' multicast addresses. If any server hearing this query can respond, it does so directly to C. If C hears no responses for some





MASH client with inner and outer searching rings



MASH subgroup with root server (solid) and subordinate servers (hollow)

Fig 3.3 Expanding-wheel topology

time, it will increase its TTL and retransmit its query again on the global multicast address. In our example, the rebroadcast query will reach root servers X, Y, and Z (the dashed, outer circle). Y and Z will ignore the retransmitted query (by checking a record of recent queries) but X will respond in the same manner that Y and Z did the first time. The advantage of this method over an expanding-disc search is that on TTL-expansion, the multicast infrastructure will have to carry the unneeded retransmission only to Y and Z. In an expanding-disc, however, the retransmission would go to all of Y's and Z's members (all previously reached destinations).

Under significant scaling conditions, expanding-wheel can lessen the multicast traffic load tremendously. This reduction is due to the fact that the majority of servers are children who do not even subscribe to G_{global}, and thus are never involved in the multicast distribution tree for the repeated queries.

Another contribution of this work is a group management protocol to form hierarchical groups automatically, where higher-level servers are "parents" of the lower-level servers, the "children." Refer to [Rosenstein *et al.*1997] for more details.

Revere does not have mandatory requirements concerning what service is used to discover existing Revere nodes. Every exemplary method discussed above can be used. Revere leaves this decision up to each individual node.

3.2.2 Three-Way-Handshake Protocol

After locating some existing Revere nodes, a new node can then negotiate with those nodes to attach itself to some of them as a new child. The negotiation between a potential child and a potential parent is a reciprocal selection procedure. An existing node needs to determine whether it wants to add the new node as a child. The new node, on the other hand, needs to determine whether it wants the existing node to be its parent.

The negotiation is handled by a three-way-handshake protocol, as shown in Figure 3.4. A potential child first sends an attach request to a potential parent. The potential parent decides whether to adopt the applicant as a new child, and sends back a reply message. The child adoption decision is machine-specific: some machines may only check to see if they have reached the maximum number of children that can be accommodated; some machines may check for more information before they make a

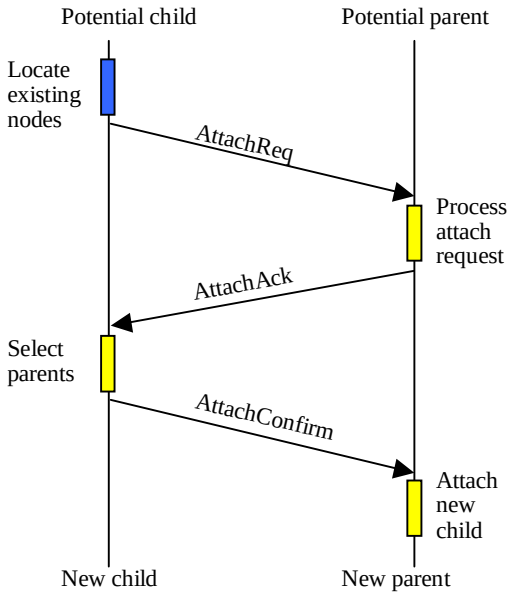


Fig 3.4 Three-way-handshake during join procedure

decision. Revere allows pluggable machine-specific child-adoption modules. For example, because it is prone to disconnection, a mobile node may choose only to be a leaf node on any RBone, without accepting any attach requests. Or a multicast-capable node may prefer nodes that can hear multicast messages as well, allowing it to maintain a single multicast IP address that reaches multiple child nodes.

If the potential parent agrees to add the applicant as a new child, it will add the applicant as a pending child and reply with an attach acknowledgement message to indicate the approval (otherwise, it will send back a negative acknowledgement message). The pending child is also bound with a timer, requiring that a confirmation message be received from the pending child within a specific period.

Upon the receipt of a positive attach acknowledgement message from the potential parent, the new node decides whether accepting this potential parent will improve its resiliency and efficiency for receiving security updates. If so, it will accept the parent,

possibly “divorcing” an existing parent. This decision procedure will be discussed further in Section 3.2.4. If the node decides to add this new parent, it will send back a confirmation message. When received, the parent will convert the requesting node from a pending child to a regular child.

During the three-way-handshake procedure, the transmission mechanism that the new parent uses to forward security updates can also be negotiated. The positive acknowledgement can contain an ordered list of transmission mechanisms preferred by the potential parent, and the confirmation message can carry the transmission mechanism selected by the child.

A potential parent can also assist in finding other parents by randomly choosing one or more of its current children and including them in its acknowledgement messages. The potential child is free to contact these nodes or ignore them, depending on its selection criteria.

Loss or transmission errors may happen for messages used in the three-way-handshake procedure. A Revere node relies on timers to handle this. For example, after an AttachReq is sent, if an AttachAck from the other side is not received in time, this node can treat the AttachReq message as lost or caught in error, or treat the other side as dead or non-existing. Similarly, after a positive AttachAck message (or an AttachConfirm message) is sent from a node, the node will begin waiting for an AttachConfirm (or HeartBeatFromChild message) from the other side. We will illustrate this in more detail when discussing the data structures in Section 3.4.

3.2.3 The Lifetime of a Join Procedure

A new node may need to continuously search for candidate parents until it finishes the join procedure. A node is allowed to simultaneously negotiate with multiple potential

parents. After a new node attaches itself to its first parent, the new node is then already a Revere node that belongs to an RBone, and is thus reachable by security updates through this parent (if nothing goes wrong). However, a join procedure will not end until all of the following conditions are met:

- The new node has attached itself to some pre-defined minimum number of parents.
- The estimated security update delivery latency from the dissemination center is faster than a pre-defined minimum value.
- The resiliency on receiving security updates is stronger than a pre-defined minimum value.

Figure 3.5 shows that a new node X contacts four potential parents: 1, 2, 3, and 4.

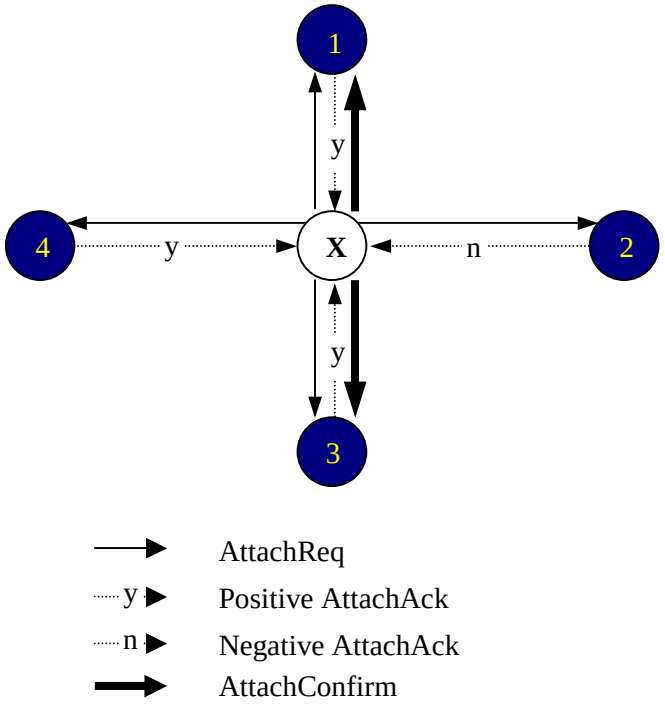


Fig 3.5 A new node's three-way-handshake
with four existing Revere nodes

After sending AttachReq messages to them, nodes 1, 3, and 4 send back positive AttackAck messages, but node 2 denies node X's attachment request. Node X selects nodes 1 and 3 to be its parents and sends back an AttackConfirm message. As a result, two new parent-child relationships are established.

3.2.4 Parent Selection

A new node typically needs multiple parents. Ideally, a Revere node should select several parents from all available Revere nodes, such that one parent provides the fastest security update delivery and the other parents offer suitable resiliency. Good parent selection, therefore, is a necessity.

To describe parent selection, we first introduce the *path vector* concept. The path vector of a node describes the characteristics of the *fastest* path for delivering a security update from a dissemination center to this node. We also call it *node path vector,* or **NPV**. It has two important parameters: a latency value and an ordered list of nodes to cross (including the dissemination center and the destination node). In the following we use *npv*(*n*) to denote node *n*'s NPV.

Similarly, we also introduce *parent path vector*, or **PPV**, to describe the path vector of the fastest path on which parent *p* is the last hop in forwarding security updates to *n*. In the following we use *ppv*(*n*, *p*) to denote node *n*'s PPV that is associated with parent *p*.

From the definitions of NPV and PPV, *ppv*(*n*, *p*) is the concatenation of *npv*(*p*) and the link connecting *p* to *n*, and *npv*(*n*) is the fastest *ppv*(*n*, *p*) among all *p*'s. For example, in Figure 3.6, *npv*(5) is the same as *ppv*(5, 3), with a 60-millisecond latency, and crosses nodes 0, 1, 3, and 5; but *ppv*(5, 4) has a 280-millisecond latency, and crosses nodes 0, 2, 4, and 5. (Recollect that all path operations are performed at the Revere level, so these path vectors do not include intermediate routers not running Revere.)

We determine the resiliency level that a parent provides by comparing this parent's PPV and the node's NPV. For instance, at node 5 in Figure 3.6, parent 4's resiliency is calculated by comparing *ppv*(5, 4) with *npv*(5) (i.e., *ppv*(5, 3)). We adopt a simple calculation by comparing the number of overlapping intermediate nodes between the two paths (the strongest resiliency level is thus 0). In our example, the resiliency level of parent 4 is 0.

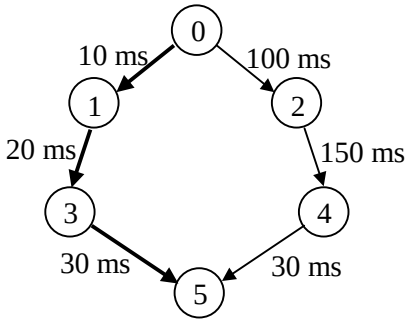


Fig 3.6 Path vector at node 5

With the notion of path vector, a child *c* selects its parents as follows. First, a potential parent *x* includes its NPV *npv*(*x*) in the positive acknowledgement message that it sends to *c*. Second, node *c* evaluates the latency from *x* to itself. To do this, *c* can contact an existing service (such as [Francis *et al.* 2001]). Or *c* can timestamp the attach request message and the positive acknowledgement message, estimate the round-trip time between *x* and itself, and use half of that value as the approximate latency from *x* to *c* (which can be further refined during RBone maintenance). Third, combining *npv*(*x*) and the latency information from *x* to *c*, node *c* derives *ppv*(*c*, *x*) and begins running the parent selection algorithm, by which node *c* determines whether adding *x* as a parent improves its efficiency or resiliency. The pseudo code in Figure 3.7 depicts this procedure. Note that if line 5 in Figure 3.7 is executed to replace node *c*'s NPV, the

```
Function boolean selectParent (ppv(c, x)) on node c:
    whether to select node x as a parent.
    npv(c): current path vector of node c.
1   if (npv(c) does not exist)  {/* c has no parent yet */
2       npv(c) ← ppv(c, x)
3       return true
4   } else if (ppv(c, x) is faster than npv(c)) {
        /* x improves efficiency */
5       npv(c) ← ppv(c, x);
6       return true
7   } else if  (∃ a parent m of node c, such that
                resiliency(x) is better than resiliency(m) {
        /* x improves resiliency */
8       return true
9   } else if  (c has not reached the minimum number
                of parents ) {
10      return true
11  } else {
        /* x improves neither efficiency nor resiliency */
12      return false
13  }
```

Fig 3.7 Parent selection based on path vector

resiliency of the parents of node *c* can be changed, since the resiliency level is calculated by using node *c*'s NPV.

If the efficiency and resiliency of a node have not reached certain preconfigured levels yet, the node can always contact existing Revere nodes to try to improve, even if the node has reached the maximum number of parents allowed. An existing parent may be replaced with a better-qualified parent. For example, node 5 in Figure 3.8 is configured to have at most two parents, and already has two (nodes 3 and 4), but in an attempt to improve its efficiency, node 5 still contacts node 1. In this case, node 1 will not be used to replace any existing parent of node 5 since the PPV associated with node 1, *ppv*(5, 1), carries a slower latency than *npv*(5), and is also less resilient than *ppv*(5, 4).

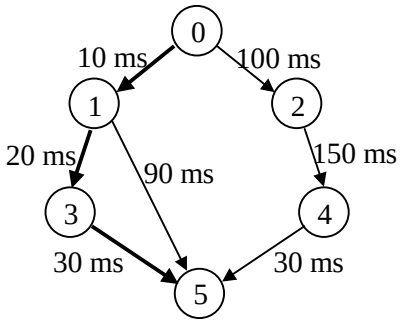


Fig 3.8 Parent selection at node 5

3.3 Adaptive Management of RBone

The changes to an RBone must be detected and quickly dealt with. Changes happen when a new Revere node joins, when an existing Revere node crashes or leaves, when one side of a parent-child link wants to untie the connection, when the characteristics of the connection between two Revere nodes changes, when a parent is detected as corrupted, when a better path is detected, or for any similar reason.

Managing an RBone is a distributed task. While an RBone can be comprised of a large number of Revere nodes, a change may only be detected by a few of those nodes. Moreover, because of the large scale of an RBone, every Revere node only has partial knowledge of the whole RBone, mostly about its neighboring Revere nodes. As a result, each Revere node has to respond to changes autonomously, thus usually asynchronously, based on its limited knowledge of the RBone.

Revere supports two different mechanisms for detecting RBone changes: explicit notification and implicit detection. With explicit notification, a Revere node can send an explicit notification message to a parent (or a child) to tear down the connection, and remove that parent (or that child) from its records at the same time. With implicit detection, a Revere node relies on heartbeat messages to detect if its parents and children are still alive. Normally, each parent periodically sends heartbeat messages to its

children, and each child periodically sends heartbeat messages to its parents. Lack of heartbeat messages for a child will eventually lead to its removal; similarly, lack of heartbeat messages from a parent will lead to the removal of the parent. Heartbeat messages can also carry timestamps to measure parent-child round-trip time. If a Revere node detects that a parent becomes distant, and thus inefficient in delivering security updates, this node can also remove that parent.

The explicit tear-down messages are in UDP format, and the delivery of those messages does not have to be guaranteed. In the case where a tear-down message is lost, the heartbeat mechanism can help. For example, if a tear-down notification from a parent to a child is lost, the parent will regard the child as already removed and stop sending heartbeat messages; although the child will still regard itself as a child of the parent for some period, lack of heartbeat messages from that parent will cause the child to remove that parent, and stop sending heartbeat messages toward that parent.

After changes are detected, some data structures must be adjusted, such as a node's NPV and PPVs. For example, if a to-be-removed parent is on the fastest delivery path to a node, the removal of that parent changes this node's NPV—that parent will no longer belong to this node's NPV. Or, if the latency from one of a node's parents is changed, this node needs to update the associated PPV. Or, if a node's NPV becomes slower than a PPV, this node needs to replace its current NPV with its currently fastest PPV.

The adjustment of data structures may propagate. For example, the update of a node's NPV can also potentially change the NPVs and PPVs of the descendents of this node. To handle this, once the NPV of a node is changed, its heartbeat messages toward its children will carry the new path vector information, allowing every child to adjust its NPV and PPVs if needed, or even choose a new parent.

The removal of a parent, either due to the lack of heartbeat messages or because of explicit notification, can cause a node to search for another parent. The node can use the same methods described in Section 3.2.1.

3.4 Messages and Data Structures

As we mentioned earlier, each Revere node relies on partial knowledge of the whole RBone to join an RBone, detect and respond to RBone changes, and receive and forward security updates. In this section we describe the key data structures relevant to RBone formation and maintenance. Because Revere is implemented in Java, an object-oriented language, in the following we describe each data structure in an object-oriented style.

Every Revere node on an RBone uses an object called "Joint" to participate in RBone formation and management. As pointed out earlier, there is an RBone that corresponds to every specific type of security updates. Table 3.1 shows the main fields of a Joint data structure.

Table 3.1: Main fields of a Joint data structure

Field	Description
parentsInfo	Information about parents of this node
childrenInfo	Information about children of this node
incomingMessages	Buffered incoming RBone messages waiting to be handled
rboneMgr	RBone communication manager responsible for sending and receiving RBone messages
topology	Topology information
searchAgent	Agent used in search for existing Revere nodes
timeoutEvents	List of events that will time out at some future time
rboneSecurityCoordinator	RBone security coordinator

A Joint object is also a Java thread. The Joint continuously processes incoming messages that are read by its associated RBone communication manager. As introduced in Section 3.2 and Section 3.3, there are three main types of RBone messages: messages used in a three-way-handshake procedure, explicit RBone maintenance messages for detaching a parent or a child, and implicit RBone maintenance messages for supporting heartbeats from a parent or a child. These messages are all derived from RBone messages, as shown in Figure 3.9. When processing an incoming message, the parent or child information may be updated, and new messages can be generated and sent.

A Joint object not only processes incoming messages, it also proactively generates new RBone messages and sends them to other nodes. Similar to incoming messages, there are also three types of such messages. First, a new Revere node needs to send AttachReq messages to some existing Revere nodes in order to attach itself to an RBone. Here, the search agent can be invoked whenever existing Revere nodes are to be found, and can be configured to use a locally preferred searching method. Second, during RBone maintenance, a Revere node needs to periodically send heartbeat messages to its

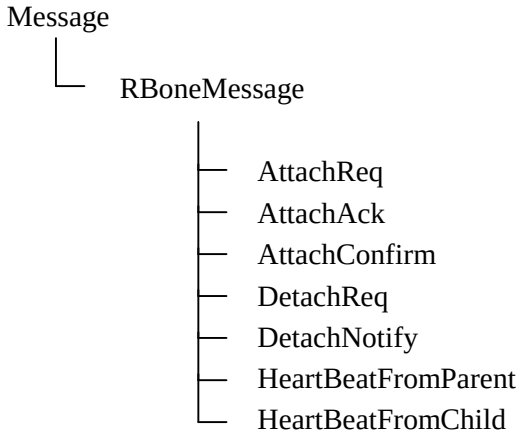


Fig 3.9 RBone messages (solid lines denote inheritance)

parents or children. Third, a Revere node may also need to send a detachment message to a parent or a child in order to specifically tear down the relationship.

A Joint object has references to parent information and child information. The *parentsInfo* object contains the list of current parents of a local Revere node and the path vector of this node. It also includes information concerning negotiations with those potential parents. AttachAck messages from those nodes that are not potential parents will simply be discarded. Similarly, the *childrenInfo* object contains the list of current children of a local Revere node. Those pending children, who have been accepted as this node's children, but have not confirmed their final willingness yet, are also kept in this object.

An RBone communication manager is responsible for sending and receiving RBone messages, and it can be configured to use either UDP or TCP for RBone message communication. An RBone communication manager is implemented as a thread, and continuously waits for incoming RBone messages at a particular port number. Support for TCP-based RBone communication involves more operations and higher cost than UDP-based communication: for every Revere node to communicate, a separate socket and an associated thread must be created. We use UDP-based RBone communication as the default.

A Joint object also maintains topology information, mainly a list of measured round-trip times between the local node and another Revere node. The round-trip time can be measured by piggybacking timestamps on proper RBone messages. The recorded round-trip time with a Revere node can also be further refined by incorporating the newly measured round-trip time with the same node. If it were available, underlying physical topology information could also be stored here.

The Joint also keeps a list of events that can happen in the future. There are two types of such events: a scheduled task to perform *at* a particular future moment, and an anticipated event to happen *before* a particular future moment.

A scheduled task is mainly the sending of a message toward another Revere node, including sending an *AttachReq* message at a future moment if a Revere node has not finished the join procedure, sending a heartbeat message toward a child, or sending a heartbeat toward a parent.

An anticipated event is mainly concerns receiving a particular type of message from another Revere node. An anticipated event can be the reception of the following messages from a Revere node:

- *AttachAck message*: After a node sends out an AttachReq message toward a potential parent, it initiates a three-way-handshake procedure. It establishes the proper negotiation state (including remembering the potential parent), sets up a timer, and begins waiting for the AttachAck message (positive or negative) from the potential parent. If an AttachAck message is not received before the timer expires, this node will destroy the negotiation state information; otherwise, this node will revoke the timer and begin processing the AttachAck message.
- *AttachConfirm message*: During the three-way-handshake procedure, if the potential parent agrees to accept the potential child, it will remember the potential child as a pending child, and send back a positive AttachAck message. The potential parent will also set up a timer and begin waiting for an AttachConfirm message from the potential child. If an AttachConfirm message is not received before the timer expires, the

potential parent will remove the pending child; otherwise, it revokes the timer and begins processing the AttachConfirm message.

- *HeartBeatFromParent message*: Corresponding to every parent, a Revere child node expects periodic heartbeat messages from this parent. A Revere node also sets a timer to handle this, typically with a time-out period that is several times the regular heartbeat period in order to be resilient to accidental heartbeat message losses. If no heartbeat message is received from a parent before the timer expires, this node will regard that parent as dead and remove it; otherwise, when a heartbeat message is received, the timer will be reset and another round of waiting for a heartbeat message will begin.
- *HeartBeatFromChild message*: This is very similar to HeartBeatFromParent message, except that a Revere node now expects heartbeats from one of its children.

Timer-controlled message communication makes an RBone resilient to message loss. As an example, after a node sends out an AttachConfirm message to another Revere node, the first node (the former) immediately records the second node (the latter) as its parent, and begins waiting for *HeartBeatFromParent* messages from the second node. If the AttachConfirm message is lost, the second node will time out on waiting for the AttachConfirm message, and remove the first node as a pending child, and certainly never issue HeartBeatFromParent messages toward the first node. The first node, without hearing *HeartBeatFromParent* messages from the second node at all, will finally time out and remove the second node as a parent.

The *rboneSecurityCoordinator* is related to securing an RBone. RBone security will be discussed in Chapter 5 in more detail.

3.5 Building a Common RBone

So far in our discussion, corresponding to every particular type of security update, there is an RBone rooted at a specific dissemination center. Different RBones are independent of one another and every RBone is an autonomous system, formed and maintained by itself.

This scales well when a node only needs to listen to several types of security updates. However, when there are many types of different security updates to listen to, a node may become overloaded. For a Revere node that is interested in *n* types of security updates, it has to join all *n* type-specific RBones.

In this section, we discuss two alternatives for designing a common RBone for delivering multiple types of security updates. Instead of being rooted at a dissemination center, a common RBone is rooted at a single rendezvous point, or rooted at multiple rendezvous points.

3.5.1 Common RBone Rooted at a Single Rendezvous Point

A common RBone rooted at a rendezvous point is very similar to an RBone rooted at a dissemination center, as described earlier in this chapter. Such a common RBone is still a self-organized overlay network with an open subscription paradigm. However, every dissemination center now has a link toward the rendezvous point. Figure 3.10 shows a sample RBone with a single rendezvous point connected by three dissemination centers, where each center is responsible for delivering some type of security update.

A common RBone rooted at a single rendezvous point has the following implications:

- The path vector of every Revere node will start from the rendezvous point.

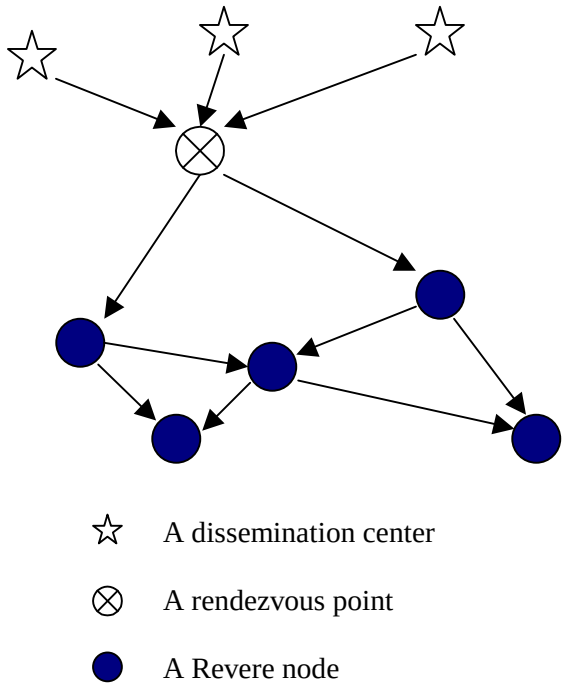


Fig 3.10 A common RBone based on a single rendezvous point

- During the join procedure of a new node, any dissemination center will not respond to an attachment request; instead, the new node can contact the rendezvous point to start a three-way-handshake procedure.
- When a dissemination center needs to deliver a security update to millions of recipients, it first sends the security update toward the rendezvous point. When a rendezvous point receives the security update, it then further disseminates the security update.
- A Revere node may now receive a specific type of security updates that it did not subscribe to. In other words, this node is simply a forwarding node for those security updates. (In fact, the type-specific RBone discussed in previous sections could also have such issues, namely some

Revere nodes may receive unneeded security updates, depending on what a dissemination center delivers.) In this case, the node will still run duplicate checking to help avoid dissemination loops. The security check, however, is optional since this node will not use those updates at all; it can rely on other nodes that really need those security updates to authenticate them. We will discuss duplicate checking and security checking further in Chapter 4 when we discuss the dissemination procedure.

A common RBone rooted at a rendezvous point also introduces new issues regarding resiliency: if the rendezvous point is crashed or subverted, or if the path from a dissemination center to the rendezvous point is broken, security update dissemination will fail. This can be addressed from two aspects. First, a rendezvous point, as a critical resource, must be protected in the same manner as that of a dissemination center. Second, the path from a dissemination center to a rendezvous point should be carefully monitored. If there is any problem with the path, it must be solved immediately.

3.5.2 Common RBone Rooted at Multiple Rendezvous Points

Instead of having a single rendezvous point, multiple equally capable and resilient rendezvous points can be established, probably sparsely distributed over the network. Every dissemination center has a link toward every rendezvous point. A sample RBone with multiple rendezvous points is shown in Figure 3.11.

Such a common RBone has the following implications:

- The path vector of every Revere node will start from one of the rendezvous points.
- During its join procedure, a new node can contact one of the rendezvous points to start a three-way-handshake procedure.

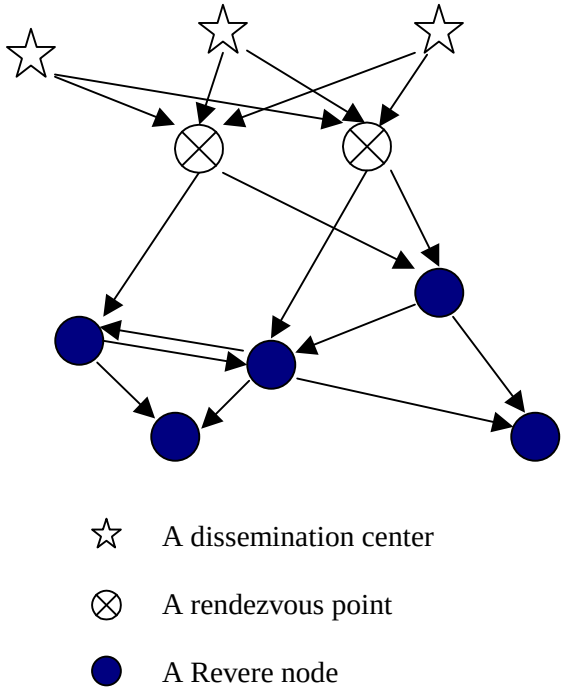


Fig 3.11 A common RBone based on multiple rendezvous points

- When a dissemination center needs to deliver a security update to millions of recipients, it first sends the security update toward all rendezvous points. When a rendezvous point receives the security update, it then further disseminates the security update.
- Just as with a common RBone rooted at a single rendezvous point, a Revere node may receive security updates it does not need. The related discussion in Section 3.5.1 applies here, too.

With multiple rendezvous points, a common RBone becomes more resilient. An attacker must compromise all the rendezvous points in order to stop the initiation of security update dissemination.

3.6 Conclusions

The Internet provides many different kinds of message transmission services, from the lower-level best-effort message delivery to the higher-level reliable message transmission. However, security update dissemination poses a unique challenge to sending a moderate amount of information to millions of recipients. This chapter shows that by forming an overlay network of those recipients at the application level, tremendous communication power and flexibility can be achieved, while successfully addressing challenges such as scalability and resiliency.

RBone, as such an overlay network, supports an open subscription paradigm. A new node that wants to receive a specific type of security update can simply attach itself to some existing nodes on the RBone for this type of security update. By proposing a three-way-handshake procedure, this chapter illustrates that a join operation can be simple but effective. RBone maintenance, which is designed to handle departures, disconnectedness, crashes, corruption or other changes to Revere nodes, ensures that an RBone is resistant to unpleasant problems while still maintaining valuable features such as scalability and resiliency. We also demonstrate that self-organization is not only flexible, but also powerful.

A highlight that comes with RBone self-organization is that every node can choose its own mechanisms for many aspects of communication, including the minimum number of parents to have, the maximum number of children to have, the preferred transmission mechanism for forwarding security updates to a child, and so on. This chapter also describes an important discretionary decision-making procedure: how to select a parent. With successful parent-selection algorithms, not only is the efficiency of receiving security updates considered, but also the resiliency of a node to receive those updates.

CHAPTER 4

Dissemination Procedure

Revere supports a dual mechanism for delivering security updates: pushing and pulling. Using pushing, a dissemination center can broadcast a security update via an RBone to all connected nodes. Using pulling, an individual Revere node can request security updates. Pushing is the main delivery method. Pulling, on the other hand, allows a node to catch up with any missed security updates.

Although pushing and pulling are used in combination in the real world, for clarity, we will discuss pushing and pulling separately in this chapter. This chapter is organized as follows. First, we discuss the dissemination principle and its relationship with other software components in Section 4.1. Following that, the role and properties of a dissemination center are discussed in Section 4.2, and the format of security updates is explained in Section 4.3. Section 4.4 begins the discussion of pushing operations, where the receiving, processing and forwarding of a security update at each hop will be illustrated. Section 4.5 describes the pulling operation, where we explain why a repository-server-based pulling mechanism is needed and show how it works. Finally, we raise some open issues and conclude this chapter.

4.1 Dissemination Principle

4.1.1 The Scope of Revere

In terms of disseminating security updates, what does Revere do and what does Revere not do?

First, it is acceptable for a Revere node to receive multiple copies of the same security update. Because of the small size, relatively low frequency and critical importance of security updates, Revere can afford to deliver a copy of a security update from every parent to every child, assuming no failures. As we saw in Chapter 3, every Revere node on an RBone is allowed to have multiple delivery paths. It is not only worthwhile, but also important to "waste" some bandwidth to ensure delivery.

Second, no strict global reliability is provided for delivering every security update to every recipient. Perfect delivery to *all* nodes is often not vital, although a security update should be delivered to at least a high percentage of all nodes. During a pushing session of security updates, neither the dissemination center nor any third party is responsible for providing reliable delivery. Even though redundant delivery is supported, pushing is still a best-effort operation. Revere defers the reliability provision to every individual node. It is up to an individual Revere node itself to determine how many delivery paths to obtain and maintain, to select which transmission mechanism to use for receiving security updates and which transmission mechanism to use for forwarding security updates, to verify whether it has received all the security updates, and to retrieve missed security updates.

Third, the dissemination service that Revere provides, including pushing and pulling, is a general service only for delivering information. A Revere service client may find it also needs other relevant services, such as a security update generating and reporting service by a third party, security update management at a dissemination center, or the application of newly received security updates at a Revere node. These services, however, are not part of Revere. For example, when a dissemination center (or probably actually its system administrator) has information of a newly discovered virus, it may decide to simply send information that a new virus is found, or decide to distribute the

signature of the new virus as well, or decide to also include the remedy for the virus. A recipient certainly will respond in different ways when receiving those security updates. The point here is that Revere is only responsible for delivering security updates.

4.1.2 Characteristics

During the period of disseminating a security update, what delivery characteristics are important?

To review our discussion in Chapter 1 (Section 1.4), such characteristics are *fast*, *resilient*, *scalable*, and *secure*. Since dissemination is divided into pushing and pulling, those properties should be maintained during both operations.

To be fast, in addition to designing an RBone to efficiently forward security updates, Revere tries to ensure that the processing time for a security update at each hop is minimal. Note, just as it does during a pushing session, a Revere node that pulls security updates may also need to forward a newly pulled security update to its children; therefore, fast processing is desirable for both pushing and pulling.

To be resilient (as we saw in Chapter 3), an RBone, as the channel for delivering security updates, supports resiliency through redundancy. Every node can choose to have multiple delivery paths, therefore multiple parents, to get security updates pushed throughout an RBone. From the pulling point of view, a Revere node should also be allowed to contact multiple places to obtain missed security updates.

To be scalable, those cases that are “rare” at lower scale may become common and must be handled. Node disconnection, for example, will not be rare when the number of Revere nodes increases to certain level; worse, different nodes may have different disconnection periods, and thus might incur different sets of missed security updates. In addition, scalability of Revere in terms of dissemination also concerns the speed of

processing security updates, the cost of storing relevant data structures, and the overhead of communication among Revere nodes.

To be secure, every security update must be verified upon its receipt. Corrupted security updates must not be forwarded. We defer security-related discussions to Chapter 5.

4.2 Dissemination Center

A dissemination center is responsible for disseminating security updates. In principle, anybody could set up a machine to be a dissemination center for some type of security information; however, users may not trust such a machine at all. We believe that a dissemination center should be strongly protected. We also believe that a dissemination center can quickly and reliably obtain security information that Revere subscribers want. Such a dissemination center can be much more easily trusted and accepted by Revere nodes, and can serve a great number of recipients.

A dissemination center is usually set up by a prestigious organization. For instance, a well-known anti-virus center can set up a dissemination center for new virus signature distribution, or an esteemed certificate authority may declare itself a dissemination center for sending certificate revocation lists. There could be more than one dissemination center for a specific type of security update (such as an early-warning signal). In this case, a cautious user can subscribe to multiple centers to cross check security update information.

As discussed in Chapter 3, the RBone dissemination center only needs to remember the identities of its direct children. This greatly helps the scalability, because the dissemination center then does not need to keep track of its whole associated RBone. Maintaining a certain number of children is almost a constant cost. For example, suppose

there are one million Revere nodes in an RBone, but its dissemination center only has ten children. The dissemination center then only needs to remember all of its ten children, and has no need to know the total number of Revere nodes in the RBone. Essentially, because every direct child of a dissemination center further forwards a newly received security update, it is sufficient for the center to forward a security update only to all its direct children in order to reach all Revere nodes.

Some information elements of a dissemination center are well known. For instance, the IP address of the center, the type of security updates that the center is in charge of, and even the geographical location of the center are probably publicly known. On the other hand, some information elements must be kept secret. For example, it would be disastrous if the private key of a dissemination center was disclosed.

4.3 Security Update Format

Figure 4.1 shows the format of a security update. It contains the following fields:

- **Type** A number can be used to represent a specific type, such as "new virus signature," "new intrusion pattern," "early warning signal," "security patch," and so on.
- **Seqno** The sequence number of the security update, ordered within the specified type.
- **Timestamp** The departure time of the security update from the dissemination center.

Fig 4.1 The security update format

- **Payload** The real content of the security update.
- **Signature** The signature that protects all of the above four fields. It is signed by the dissemination center.

When there are multiple centers, a security update also needs to carry the IP address of the dissemination center.

4.4 Pushing: A Store-And-Forward Mechanism

A store-and-forward mechanism is designed for the pushing operation. Via this mechanism, security updates are pushed from a dissemination center to every node on an RBone.

A pushing session begins with the composition of a security update by a dissemination center. After the payload of a security update is provided, the center adds the type and the sequence number of the security update, stamps the departure time, and signs the message with a digital signature covering all other fields of the security update. Strictly speaking, the timestamp here is not precisely the departure time, because the security update has to be signed after the timestamp is available and before the actual departure time.

After a security update is composed, the dissemination center then forwards it toward all its children on the RBone. Every child will process the security update, deliver it to the local application that needs the security update, and, if the node has any children, forward the security update to those children. This store-and-forward procedure repeats recursively at every Revere node.

The store-and-forward mechanism at each node uses two types of jobs—*input job* and *output job*, and employs one main data structure—*security update window*. Figure 4.2 shows those components at a Revere node. Both input job and output job are called

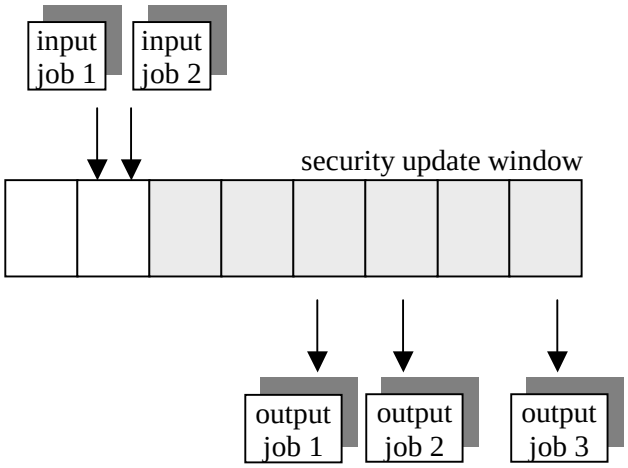


Fig 4.2 I/O jobs and security update window at a Revere node

dissemination jobs. An input job is responsible for receiving incoming security updates from parent nodes, processing them, and buffering them into the security update window. An important part of the processing is checking the authenticity of the update. An output job fetches security updates from the window and delivers them to local applications that need security updates or to child nodes on the RBone.

4.4.1 Adaptive Transmission Mechanism

Heterogeneity among RBone nodes will be common at large scale. In particular, the transmission mechanism between a parent and a child may have to be tailored to the local conditions or configurations. Revere allows an arbitrary transmission mechanism to be used when a security update is forwarded from a parent to a child, instead of enforcing any particular method for forwarding security updates from node to node. The two nodes themselves can decide which specific transmission mechanism is best for their communication.

Possible transmission mechanisms include TCP, UDP, TFTP, IP multicast, broadcast, etc. Email could be used as well. Or, in an extreme case, a floppy disk can even be used to manually transfer a security update from one machine to another. In principle, all it

takes is for the two nodes to agree on the transmission mechanism used between them. No other nodes will be affected at all.

Such an agreement can be reached during the three-way-handshake procedure when the child is trying to attach itself to the parent. The AttachReq message from a potential child should carry a list of acceptable transmission mechanisms for receiving security updates, ordered by preference. The potential parent, in addition to checking other constraints when deciding whether to adopt this potential child, would then also check the list of transmission schemes to see whether it can support at least one of them. If a particular transmission scheme can be selected, and other child-selection constraints are all met, the potential parent can then include this selected transmission scheme in the positive AttachAck message toward the potential child. In turn, the potential child will also check to determine if an acceptable transmission scheme is selected. If so, it sends back an AttachConfirm message to formally become the child of the potential parent. On the other hand, disagreement on the security update transmission mechanism can cause the three-way-handshake to fail at any point of the negotiation.

By default, Revere supports UDP-based parent-to-child security update transmission. If a potential child does not specify a list of acceptable schemes, presumably it will only accept security updates forwarded using UDP.

Given this flexibility in choosing the best transmission scheme, the two communicating nodes can select the transmission mechanism that is the most adaptive to the local communication environment, current execution context, or particular user preference (Figure 4.3). The following are several examples or guidelines for different needs:

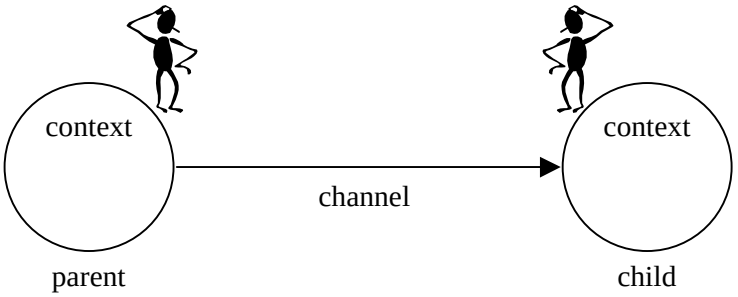


Fig 4.3 Adaptive parent-to-child security update transmission

- **Adaptation to local communication environment**

  When the channel between a parent and a child is error-prone, FEC (forward error control) or another error recovery method may need to be applied. When the channel is a low bandwidth, the security update can be compressed at the parent node, and decompressed at the child node upon receipt. When the channel is capable of broadcasting, such as Ethernet, a security update may be delivered to multiple nodes through a one-time broadcast. As another example, TCP is preferred to UDP in terms of providing reliable delivery, but TCP is more costly than UDP because it requires a Revere node to maintain a connection with every individual parent or child node.

- **Adaptation to current execution context**

  Depending on the current running environment, a Revere node might also have some particular policies regarding forwarding security updates. For example, when a Revere node is about to forward a security update, if it happens to have many children at the moment, it may prioritize its children and forward updates first to those with a high priority. On the other hand, if a node recognizes that all its parents are heavily overloaded

with children, it may search for another parent that has a lighter load, and thus perhaps obtain a faster delivery of security updates.

- **Adaptation to particular user preferences**

  Users could have their own preference, too. As a simple example, a user could specify TCP as the security update forwarding protocol. A user might prefer not to compress a security update, even when forwarding it along a low-bandwidth channel, since the compression and the decompression incur a longer processing time. Additionally, security updates are usually already of small size.

4.4.2 UDP-Based Pushing

When UDP is used to push security updates, a Revere node incurs low overhead. As a connectionless transport protocol, UDP allows a recipient to listen on a specific UDP port for messages from multiple sources. As a result, a single input job can be used to receive UDP-encapsulated security updates from all parents. Similarly, a single output job can be used to forward security updates to all children.

Figure 4.4 shows dissemination jobs and the security update window when UDP is used. It also shows the relationship between the output job of a parent node and the input job of a child node. In particular, an output job needs to communicate with multiple input jobs.

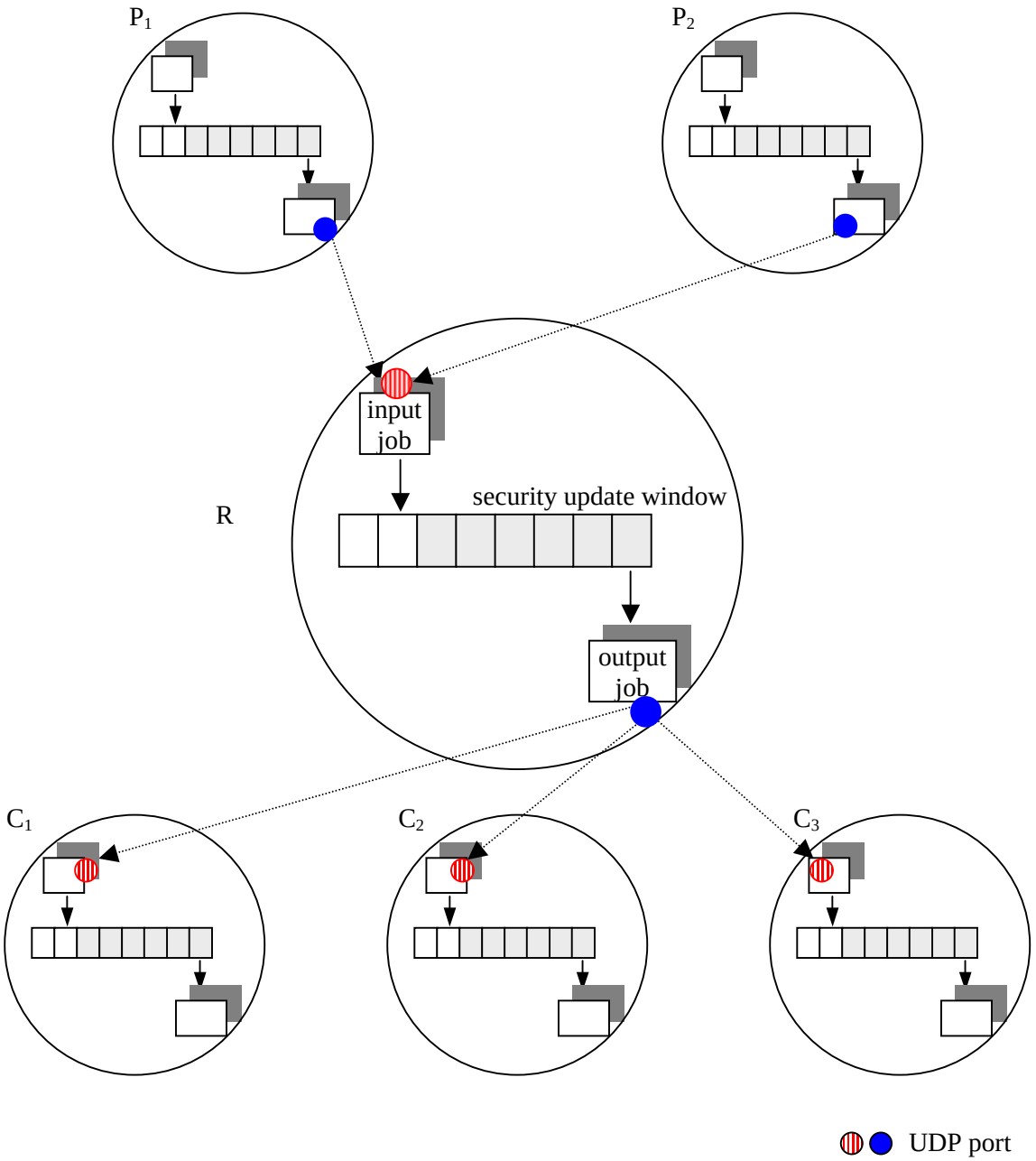


Fig 4.4 UDP-based pushing operation
(Node R has two parents: P_1 and P_2, and three children: C_1, C_2 and C_3.)

4.4.3 TCP-Based Pushing

In the case of using TCP for pushing security updates, the overhead will be higher than using UDP. TCP, as a connection-based protocol, requires both the sender and the receiver to maintain the connection. To keep listening to TCP-packet-encapsulated security updates from every source, a Revere node must maintain a separate TCP connection with that source, and read every incoming message. Since it is natural to associate each TCP connection with a separate task, and every Revere node is allowed to have multiple parents, Revere uses multiple input jobs to concurrently read security updates from all parents.

Similarly, a Revere node may have multiple children, and, corresponding to every child, a separate TCP connection is required to push security updates toward that child. Revere uses multiple output jobs to concurrently send security updates toward all children.

Therefore, a one-to-one association relationship is maintained between the output job of a parent node and the input job of a child node. Figure 4.5 shows an example of TCP-based security update pushing. Here, at node N, there are two input jobs corresponding to its two parents, and three output jobs corresponding to its three children.

TCP connections could be made on a temporary basis: a parent establishes a new TCP connection with a child for a new security update, delivers the packet representing the update, then tears down the TCP connection. However, this slows down the forwarding speed of security updates. TCP connections could also all be handled in a single input job (or a single output job), but this then requires the input job (or output job) to multiplex and synchronize among multiple TCP connections, making it a complex issue.

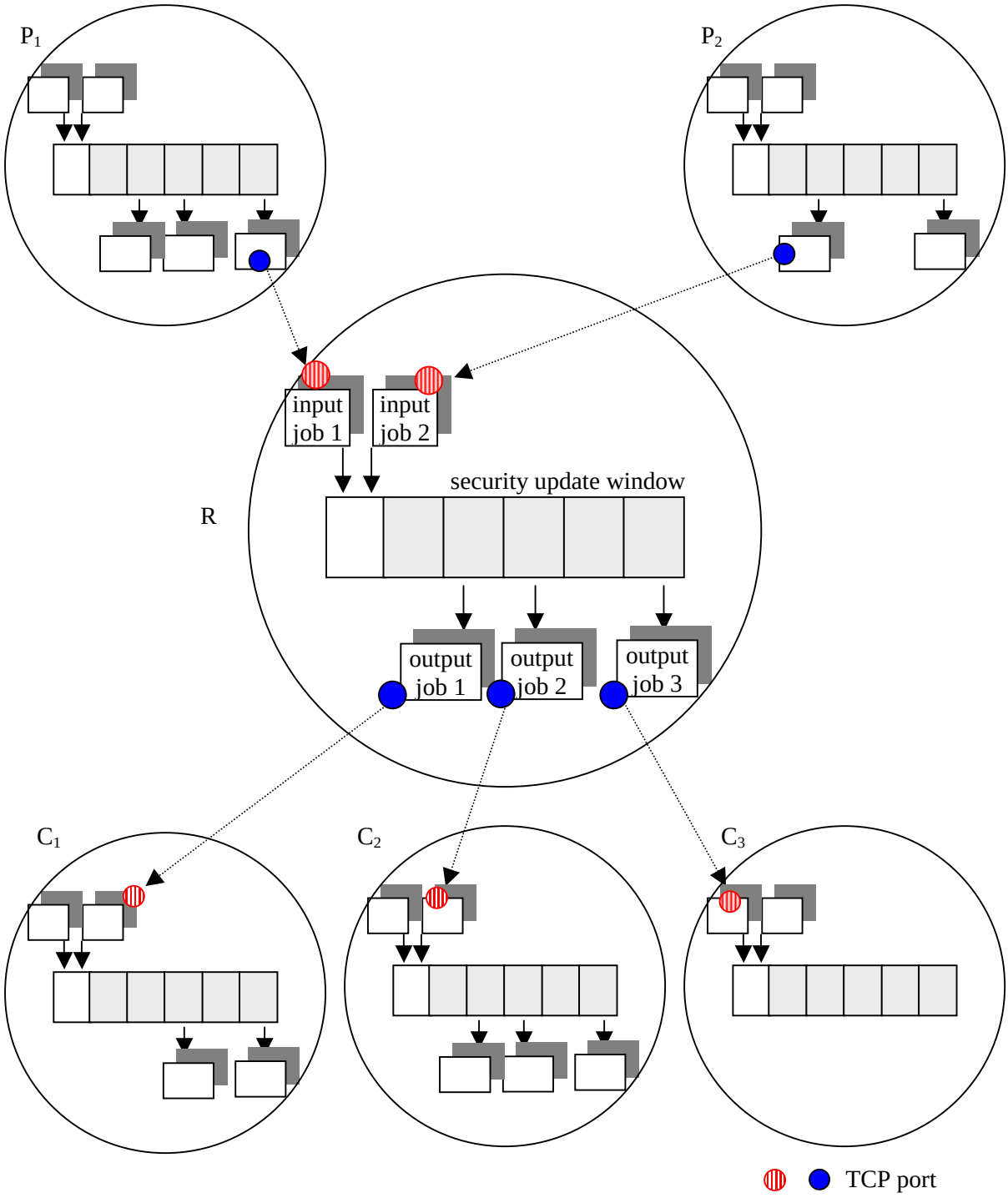


Fig 4.5 TCP-based pushing operation
(Node R has two parents: P_1 and P_2, and three children: C_1, C_2 and C_3)

4.4.4 Concurrently Running Multiple Transport Protocols

A Revere node could be running multiple protocols in receiving or forwarding security updates. For instance, a node could run UDP-based input jobs, but TCP-based output jobs. Or, the node could run one UDP-based input job to collectively receive security updates from several parents, and run multiple TCP-based input jobs to separately receive security updates from every other individual parent. Likewise, the node could run one UDP-based output job to send security updates to some of its children, and run a TCP-based output job to send security updates toward every other individual child.

Although TCP and UDP are the two most commonly used transport protocols, Revere does not exclude other transmission primitives from being used for forwarding security updates. Again, this can be negotiated when a node tries to attach itself to another Revere node as a child.

4.4.5 Bootstrap of Dissemination Jobs

Dissemination jobs, namely input and output jobs, can be started immediately when a node becomes a child of another (*immediate bootstrap*), or started when a parent node is about to forward a security update to a child node (*on-demand bootstrap*). The former ensures that those necessary dissemination jobs are ready when a security update needs to be forwarded; the latter helps save some cost by starting them on-demand.

The bootstrap of dissemination jobs has three main components: output job bootstrap at the parent node, input job bootstrap at the child node, and a necessary information setup.

The bootstrap of an output job is straightforward. In both immediate bootstrap and on-demand bootstrap, the parent node will first check to see whether a new output job

must be created and started. For example, when UDP is used, it is not necessary to start a new output job, unless there is no UDP output job yet at all. However, when TCP is to be used, a new output job must be created and started that corresponds to the new child. Step 1 in Figure 4.6 corresponds to the bootstrap of an output job.

The bootstrap of an input job is handled by a node I/O manager. A node I/O manager is a daemon process that handles two types of messages: LinkInit and LinkInitAck. The output job of the parent, which could be just created as described in the previous paragraph, will send a LinkInit message toward the node I/O manager of the child node (Step 2 in Figure 4.6). Upon receipt of the LinkInit, the node I/O manager at the child node will check to see if a new input job must be created; if so, a new input job will be created and started (as shown in step 3 in Figure 4.6). Similar to the creation of an output

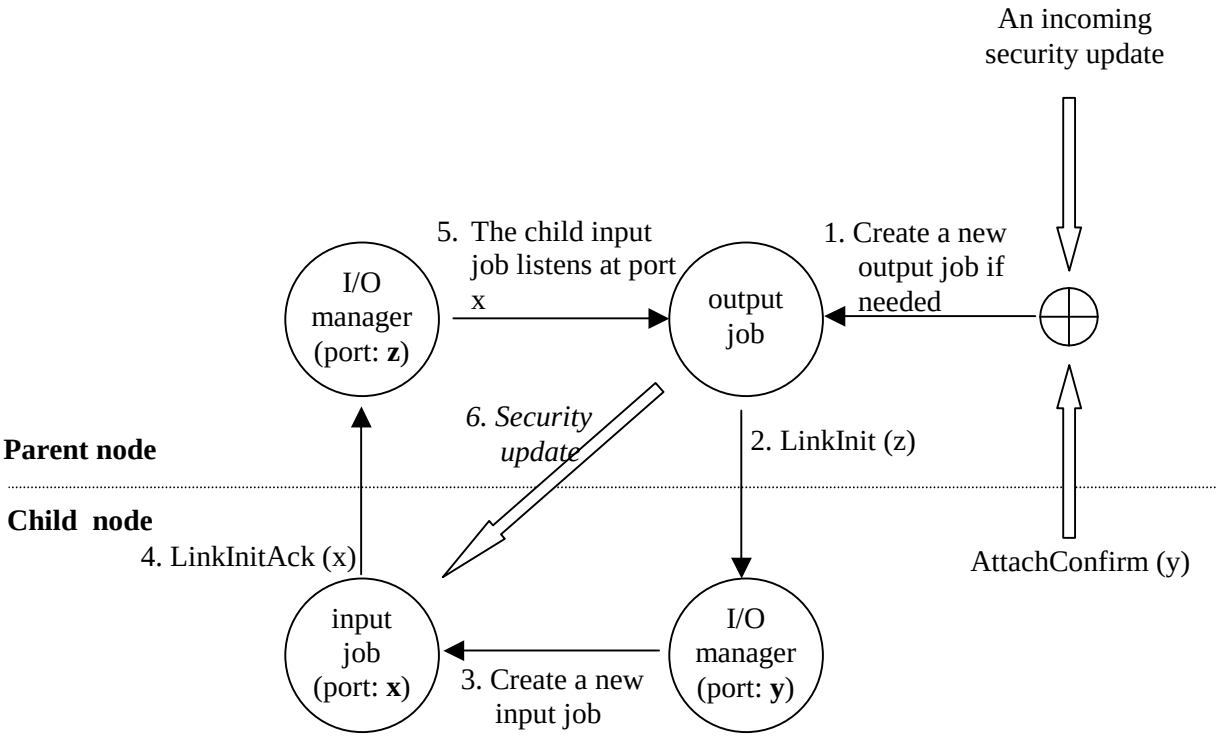


Fig 4.6 Bootstrap of dissemination jobs

job, UDP-based pushing will only need one input job per node, but TCP-based pushing will need one input job per parent.

After both the output job at the parent and the input job at the child are created, some necessary information must be set up as well, depending on what transmission mechanism will be used to forward security updates. In particular, it may be necessary to provide the output job with information regarding the input job at the child. For instance, when UDP is used, the port number on which the input job of the child listens to incoming security updates may not be known to the output job at the parent. Revere allows the input job at the child to send a *LinkInitAck* message toward the node I/O manager at the parent node, which contains necessary information of the input job at the child (Step 4 in Figure 4.6). Upon receipt of a LinkInitAck message, the node I/O manager at the parent node will notify the output job of such information (Step 5 in Figure 4.6).

Revere allows a node I/O manager to sit on an arbitrary port number and piggyback the port number information of a node I/O manager in other messages, as shown in Figure 4.6. For instance, based on AttachConfirm messages during a three-way-handshake procedure, the output job knows the port number of the node I/O manager at the child node. In turn, based on LinkInit message, the child node knows the port number of the node I/O manager at the parent node.

After the bootstrap of dissemination jobs, security updates can be forwarded from the output job at a parent node to the input job at a child node (Step 6 in Figure 4.6).

4.4.6 Processing Security Updates

A security update must be processed before it is forwarded to another Revere node. Revere is a high fan-out delivery service. It must ensure that a normal node will not

forward security updates that are erroneous, corrupted, or duplicated (including maliciously replayed).

Through input jobs, Revere mainly performs two checks in terms of processing a security update: duplicate checking and security checking. Only after a security update has passed both checks can it be stored in the security update window at a Revere node. Figure 4.7 shows the procedure of processing a security update.

Duplicate checking. Because of the redundancy built into an RBone, nodes typically receive duplicate copies of each security update. Duplicate copies are identified by the sequence numbers carried in security updates and will be dropped, rather than put into the security update window. In addition to preventing local reuse and retransmission to children, this mechanism avoids dissemination loops.

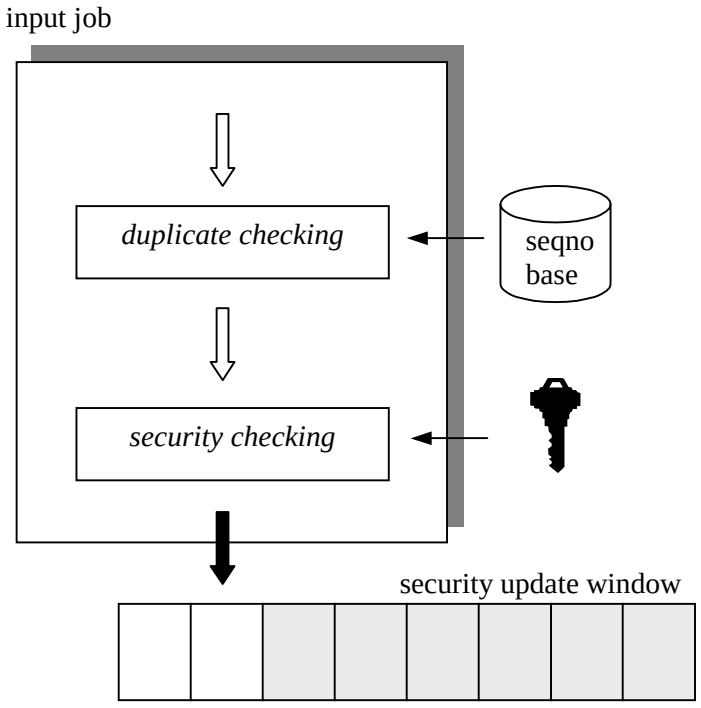


Fig 4.7 Security update processing by an input job

Duplicate checking is a lightweight operation. For every type of security update, a Revere node maintains a sequence number record of historical security updates, and duplicate checking is done to essentially compare the sequence number of a current security update against this record. The record is the range of historical sequence numbers. Given that sequence numbers are usually continuous, it is easy to maintain a complete list of all updates that a node has seen.

Duplicate checking also helps prevent replay attack. With a small processing overhead, a replayed security update can be easily filtered out.

Security checking. A newly received security update may contain a transmission error, or worse, it may have been corrupted. An attack may also inject illegitimate security updates, probably by impersonating a dissemination center. The input job must verify every security update to ensure its authenticity and integrity before buffering it into a security update window. Details will be discussed in Chapter 5.

After passing duplicate checking and security checking, a security update will then be accepted and stored in the security update window as a unique authentic security update.

4.4.7 Security Update Window

The security update window is a place where accepted security updates are stored and fetched. An input job views a security update window as a queue, and tries to append every newly accepted security update to the tail of the queue. An output job also views it as a queue, and tries to fetch security updates, one by one, from the head of the queue.

As we described earlier, a security update window may be simultaneously accessed by one or more input jobs and one or more output jobs. As a result, the security update window must be synchronized. On one hand, if there are multiple input jobs in a Revere node and each of them receives a copy of the same security update, they may all believe

that a unique authentic security update has just been received (after processing the corresponding copy). They must be synchronized so that one, and only one, copy is buffered into the security update window. On the other hand, a security update should not be removed from the security update window until all the output jobs have fetched it.

4.5 Pulling Security Updates

4.5.1 Problem Statement

When security updates are pushed from a dissemination center, some nodes may not be connected or may be temporarily turned off. When they regain connectivity, they will want to receive the missed security updates. However, this is not easy for the following reasons: (1) their parents may not have retained all missed copies; (2) depending on the length of the disconnection period, those original parents may not be parents any more (lack of heartbeat messages from a disconnected child will eventually cause that child to be removed); (3) even if a once disconnected node has not missed any security updates at all, it wants to assure itself of this fact. In the third case, a reconnected node can ask a parent, since both record the historical sequence numbers, but this may fail if the parent-child relationship is already torn down.

4.5.2 Possible Approaches

One approach to the missed-update problem is to use a reliable transmission mechanism. However, although some reliable transmission mechanisms, such as TCP, can endure a short disconnection, a longer disconnection period will result in a timeout of the transmission, causing a node to miss security updates. This also restricts Revere nodes from using other transmission mechanisms, such as UDP.

Another approach is to rely on the dissemination center to retransmit those missed security updates. This is a daunting task in that an RBone is typically of large scale, and

it is difficult to determine when retransmissions have successfully reached all nodes. Worse, different nodes usually have different disconnection periods; this implies every node may have a different set of missed security updates, further complicating this retransmission solution.

A third approach is to let a dissemination center periodically rebroadcast old security updates along an RBone, making it likely that sooner or later all Revere nodes (including those reconnected) will receive a particular update. However, since every Revere node runs duplicate check before forwarding a update, an update that is rebroadcast may not travel very far in an RBone.

4.5.3 Our Approach—Repository Server Query

A more general solution is to have disconnected nodes inquire about security updates that occurred during disconnections. Revere designates some nodes as repository servers for storing old security updates and for responding to inquiries.

In Figure 4.8, when node A reconnects to the network, it contacts repository servers R_2 and R_3 to see whether it has missed some security updates; if so, it will pull the missed updates from R_2 or R_3. Furthermore, if node A has children, it will immediately forward those newly pulled security updates to them.

To process a newly pulled security update, the same procedure is applied as in the pushing operation. Both duplicate check and security check are necessary steps here, as shown in Figure 4.7.

4.5.4 Selection of Repository Servers

One important issue with the repository-server-based approach is the selection of repository servers. A Revere node needs to locate some repository servers for missed security updates.

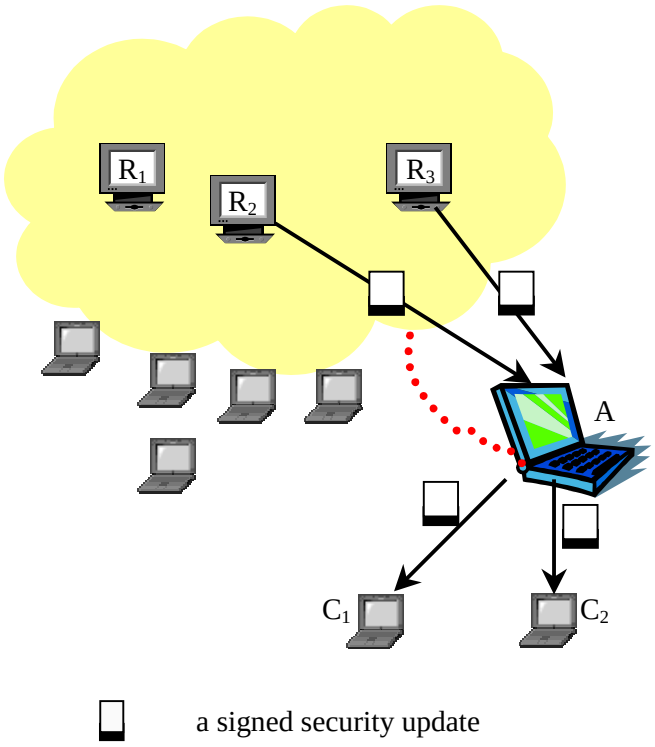


Fig 4.8 Pulling security updates from repository servers

A simple way to handle this is static configuration. A static set of repository servers can be provided when Revere is installed, where each repository server is believed to maintain stable connections to the network. Problems arise when the set of repository servers becomes dynamic. Although relatively rare, some repository servers may still fail, or some other nodes may become better suited to the role of repository server. A special security update that contains the current list of repository servers can be disseminated. However, as a one-time delivery, this list will not reach those disconnected nodes, resulting in a chicken-and-egg problem; nodes that joined an RBone after the dissemination of this security update will also miss the repository server information.

Revere employs a dynamic repository server election, maintenance and notification mechanism. First, every Revere node on an RBone is allowed to nominate itself as a

repository server, and the nomination will be received and checked by the dissemination center of the RBone. Second, an existing repository server may also fail (or decide to degrade itself into a normal Revere node), which will be detected by the dissemination center (see discussion below). Third, whenever there is a change to the set of repository servers, the dissemination center will notify Revere nodes of the change (also see discussion below).

Heartbeat messages, used for RBone maintenance as described in Chapter 3, are also used here for the above purposes. Heartbeat messages from children, propagating all the way toward a dissemination center, are employed to report the addition and subtraction of repository servers. Heartbeat messages from parents, originating from a dissemination center, are employed to indicate those up-to-date repository servers.

In detail, when a Revere node nominates itself as a repository server candidate, it will add itself to the repository candidate list in the heartbeat messages toward every parent. In turn, when a parent receives repository candidate lists from its children, it will aggregate those lists to generate a new candidate list, and piggyback the new list on its own heartbeat message toward its own parent. This repeats until the dissemination center receives a final list of all repository candidates.

The dissemination center will check the list of the repository candidates and select some of them to be the repository servers for the whole RBone. The center will then propagate the selection results through heartbeat messages toward its children, together with the list of already existing repository servers. Every child will record the new list of repository servers and also forward the information toward its own children, again through heartbeat messages.

Although repository servers should rarely be unavailable, Revere is still designed to be capable of handling failures of repository servers. In addition to the repository

candidate list, heartbeat messages from a child can also carry the identities of current repository servers. Similarly, a parent can aggregate such information carried in the heartbeat messages from all of its children, and piggyback the aggregated result on its own heartbeat message toward its own parent. However, the heartbeat messages clearly will not be generated and reported from a failed repository server; as a result, the final list of current repository servers that is received at a dissemination center will not include the failed repository server.

4.5.5 Contacting Repository Servers

Since each Revere node keeps a local list of available repository servers, it then can make an inquiry to one of the repository servers about missed security updates.

A problem we need to solve here concerns the reliability of the pulling operation. A repository server receiving an inquiry might very well be subverted itself, and thus fail to deliver some of the security updates received. The security update authentication mechanism (to be discussed in Chapter 5) ensures that a subverted repository server responding to an inquiry cannot forge false security updates, but the subverted repository could easily fail to deliver some of the updates it had received.

A Revere node can obtain some degree of certitude that *all* security updates that were missed during disconnection have been retrieved. To do so, Revere again employs redundancy to achieve high assurance. As we described earlier, Revere builds multiple repository servers, and a Revere node can just contact more than one repository server to obtain missed security updates. Or, instead of literally pulling security update copies from each of repository servers contacted, a node can just pull security updates from only one of them—a “master” repository server, and contact other “slave” repository servers to check on whether the master repository server provided a complete set of missed

security updates (typically done by comparing the range of sequence numbers of recently disseminated updates).

A key benefit of having repository servers is that they are always available for obtaining security updates. A node on an RBone can also make use of this fact. For example, if node *x* receives updates *n*, but has not received update *n-1* yet, it can always query a repository server. Or, if a node has not received any security updates from its parents in a suspiciously long time (and the length of time is at the node's own discretion), it can check with a repository server.

This characteristic offers some protection against the possibility that all of a node's parents could be corrupted. If the corrupted parents fail to ever send updates, the node can uncover the problem by checking with the repository servers. The node can then choose new parents, provided, of course, that it makes sure the chosen repository servers are not its parents.

4.6 Open Issues

One issue we have not looked into is adaptive redundancy. Adaptive redundancy has two aspects to consider: (1) a Revere node may switch to different parents; (2) a Revere node, which initially chose to have *n* parents, may decide to have $n+\Delta$ parents if there is a need to increase the level of redundancy, or $n-\Delta$ parents if a lesser number of parents can provide satisfactory certitude of security update delivery. How a Revere node chooses the initial value of *n* and the value of Δ, based on node policy and past observation, deserves more research.

As for security checking during security update dissemination, Revere encloses a digital signature on every security update. Since verification of a digital signature is

typically slow, other verification mechanisms have been studied. Further investigation of these mechanisms is needed.

"Virtual child" is another concept that could lead to some open issues. Normally, a Revere node can have multiple children, where every child is physically a node. However, one could define a collection of nodes as a single "virtual" child, with just one single IP address. For example, such a virtual child can simply be a group of IP-multicast-capable nodes, and a single IP multicast address is recorded at the node as a single child. Such a virtual child can also be a subnet's broadcast address, which can be used to reach all Revere nodes on the subnet. Although this concept improves the efficiency of forwarding security updates, how to enroll or dismiss a virtual child is an issue to investigate.

Another issue concerns the selection of a repository server to contact. While a node keeps a list of repository servers available, some of them are deemed to be topologically closer than others, and it is desirable for the node to find out which one to contact for fast delivery of missed security updates (if any). One might think that the delivery performance here for a reconnected node is not important, since the node has been disconnected for an arbitrarily long period. This is not correct. It is the vulnerable period that a node really cares about. Using a security update regarding a new network-based attack as an example, a disconnected node is resistant to the attack, and the disconnection period is not counted as part of vulnerable period at all; however, whenever the node reconnects, it becomes vulnerable to the attack, and its vulnerable period immediately starts. So, it is important that a reconnected node receive missed security updates as quickly as possible.

When a node tries to contact multiple repository servers, a question arises as to what combination of repository servers would provide the best resiliency. On one hand, the

delivery of missed security updates does not have to rely on the associated RBone—a direct point-to-point delivery can be used. On the other hand, using RBone may possibly provide better disjointness, since an RBone itself builds rich redundancy.

4.7 Conclusions

In this chapter we described a dual mechanism for disseminating security updates—pushing and pulling. Pushing relies on an RBone to forward security updates (initiated from a dissemination center) from node to node. Pulling relies on repository servers embedded within the network, and Revere nodes can query them on missed security updates.

Both pushing and pulling are resilient operations. Since a Revere node on an RBone can choose to have multiple delivery paths (as discussed in Chapter 3), the node could receive multiple copies of a security update along multiple delivery paths during a pushing operation. The node could also contact multiple repository servers to resiliently obtain missed security updates (if any are missed) during a pulling operation.

Both pushing and pulling are adaptive. During pushing, a node could adaptively choose the best suitable transmission mechanism to forward security updates to its children. During pulling, a node could contact *n* repository servers at its own discretion.

Both pushing and pulling are based on a structure that is dynamically maintained. An RBone used for pushing operations is self-organized, such that the failure or corruption of any Revere node can be detected. Repository servers used as the sources for pulling operations are also monitored so that every node can keep an up-to-date list of repository servers available.

Both pushing and pulling are lightweight, using the same procedure to process a security update, which includes duplicate check and security check. This helps promote fast delivery of security updates.

Both pushing and pulling are scalable. During the pushing operation on top of an RBone structure, every Revere node only needs to forward a security update to its own children. Unless a node has an undue number of children, this involves a very small amount of computation and communication cost. During a pulling operation between a repository server and a Revere node, neither incurs a significant amount of effort, unless a very limited number of repository servers are serving a huge number of Revere nodes simultaneously.

CHAPTER 5

Security

Revere assumes that a large percentage of Revere nodes are cooperative; however, with Revere running at Internet scale, it is unrealistic to assume that no Revere nodes have been subverted. Revere, as a service for delivering security information, can be a very tempting target for attackers. If attackers can misuse or abuse Revere, they can achieve various malicious goals; for example, a corrupted Revere system may become an ideal carrier to help propagate network worms or other threats. Therefore, Revere security must be carefully addressed, including both the security of the dissemination procedure and the security of RBone management.

In this chapter, we first discuss the security of the dissemination procedure, including integrity, authenticity and availability of security updates, replay prevention, revocation of the public key of a dissemination center, etc. We then address the security of Revere overlay networks, where we introduce the peer-to-peer security scheme negotiation algorithm and the pluggable security box. We also look at possible attacks and their countermeasures. At the end of this chapter, we discuss open issues, such as secure monitoring of dissemination progress, intrusion detection and reaction, and denial-of-service attacks.

5.1 Assumptions

The following assumptions are made in addressing the security of Revere:

- **Any node (except dissemination centers) could be corrupted**

  Revere does not enforce any policy on new node subscription procedures. Any node, evil or angel, can join Revere if it chooses. Also, given that an RBone may include a great number of nodes, the possibility that a Revere node is subverted must not be ignored. Therefore Revere, as an open-membership system, must secure itself based on this assumption.

- **A large percentage of Revere nodes are cooperative**

  This is necessary to ensure that benign Revere nodes still have a chance to receive security updates. If every Revere node were surrounded by malicious nodes, security update delivery wouldn't succeed at all. In other words, if all parents of a Revere node were corrupted, it wouldn't be able to receive authentic security updates; or, if all children of a Revere node were corrupted, it also couldn't help deliver authentic security updates.

- **The private key of a dissemination center may be compromised**

  Although a dissemination center will be protected very carefully, Revere still acknowledges the possibility that the private key of the center might be compromised, which can cause disastrous results if not addressed.

- **The public key of a dissemination center is well known**

  The public key of a dissemination center can be configured when a user installs Revere. It can also be made available through a web site. A certificate authority may also be contacted to obtain the public key of a dissemination center. Furthermore, if the dissemination center signs a message using its private key, every Revere node will be able to verify the digital signature of this message.

- **No uniform security scheme to protect node-to-node control messages**

  Due to the large scale of Revere, an RBone may support millions of Revere nodes. It is unrealistic to assume that every node at such a scale will adopt the same security scheme, apply the same security policy, or use the same configuration. Moreover, even neighboring nodes on an RBone may appear to be in different domains, thus possibly enforcing different security schemes.

- **The set of security schemes supported by different Revere nodes may overlap**

  Although a Revere node may run different security schemes from other Revere nodes, there may be one or more security schemes that both this node and another node support, making it possible that the two nodes can agree on some security schemes to protect their communication.

5.2 Security of Dissemination Procedure

5.2.1 Objectives and Requirements

The objectives of securing the dissemination process include the following:

- **Integrity of security updates**

  While a security update is being delivered to Revere nodes, it may be corrupted by malicious nodes on delivery paths. Data errors can also be introduced due to transmission problems. Any modification, whether intended or not, must be detectable. The integrity of security updates must be guaranteed.

- **Authenticity of security updates**

  Security updates originate from a dissemination center. However, an attacker may be able to forge security updates to fool Revere nodes,

probably by impersonating a dissemination center. Every node should be able to verify whether a security update has indeed originated from a dissemination center, and a security update forged by an attacker should not be accepted. Revere must quickly detect the forgery.

- **Availability of security updates**

  A new security update can be suppressed or misdirected, thus failing to reach its destinations. As discussed in Chapter 2, encryption cannot help at all, since an encrypted security update can still be suppressed, misdirected, or damaged. Authenticated acknowledgements may help, but those acknowledgments are also susceptible to interruption threats. Even if acknowledgements are successfully delivered, retransmitted security updates often use the same path and will be subject to repeated interruption threats. Revere should address these issues and achieve the best availability in delivering security updates.

- **Replay prevention**

  Old security updates may be replayed to flood an RBone. Given that those old security updates will be verified as authentic and integral, it is important that Revere not blindly forward those replayed security updates and help flood the network.

All those objectives correspond to the goal of the "security check" box in Figure 4.7 (Chapter 4). Note that Revere does not protect secrecy. In light of Revere's free subscription model, security updates are not secret information; every node is allowed to join Revere to obtain the automatic notification service. Even non-Revere nodes are free

to read and use Revere updates (if they can obtain them), though Revere does not attempt to make them available.

5.2.2 Why Public Key Cryptography

There are two main types of cryptography: asymmetric-key-based cryptography and symmetric-key-based cryptography. Revere uses asymmetric-key-based cryptography (i.e., public-key cryptography) to protect the dissemination process.

Symmetric-key-based cryptography will not work well for Revere. With a large number of machines in an RBone, symmetric-key-based cryptography does not scale. To guarantee that a security update is indeed signed by a dissemination center using symmetric cryptography, a Revere node must ensure that only the center and itself know the secret key (if a single key was shared by all Revere nodes, any Revere node could forge updates). This implies that the dissemination center must maintain a different secret key for each Revere node. Worse, since now every copy of the same security update will be signed using a different key, thus carrying a different signature, a different copy of the same security update is expected at each Revere node. As a result, the dissemination center has to unicast a different copy to every Revere node.

Instead of using different keys to sign the same security update, asymmetric-key-based cryptography allows a dissemination center to sign a security update using the same key—the dissemination center's private key. Thus, every Revere node expects to receive the same replica of the security update, and can use the public key of the dissemination center to verify its integrity.

5.2.3 Integrity of Security Updates

One method of protecting security updates from errors is to append error correction code in a security update. Unfortunately, this only helps correct transmission errors. An

attacker, while tampering with other fields of a security update, will be equally capable of modifying the error correction code to make the tampered security update appear genuine.

Instead, Revere adopts public-key cryptography to protect security updates from both malicious manipulation and transmission error (Figure 5.1). When a dissemination center is about to send out a security update, it is required to sign the security update using its private key. The format of a security update is shown in Figure 4.1 (Chapter 4), where the signature protects the type, seqno, timestamp, and payload fields of a security update.

Revere nodes use the public key of the dissemination center to authenticate security updates. When a new security update is received, a Revere node will apply the public

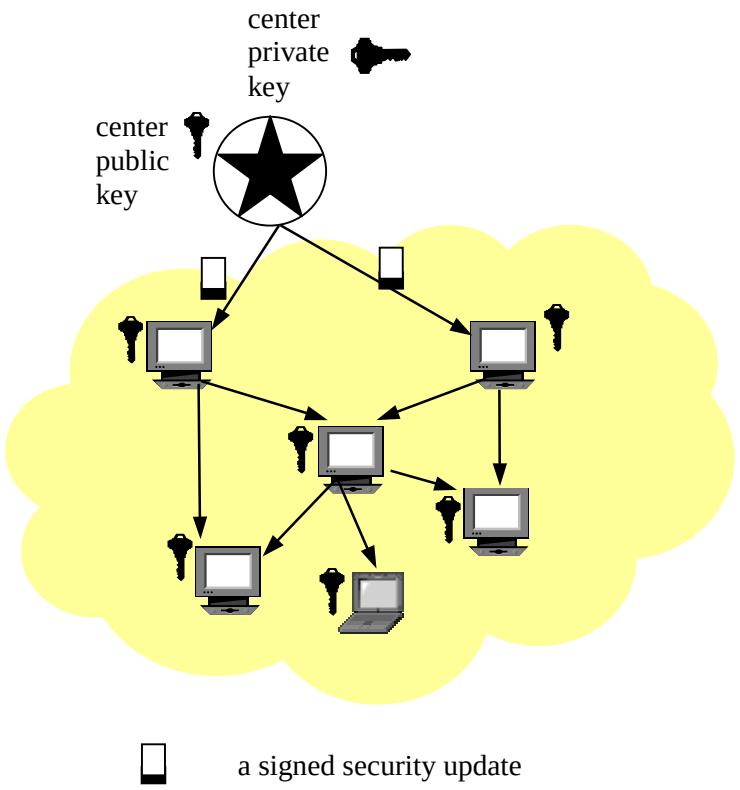


Fig 5.1 Integrity protection of a security update

key of the dissemination center to verify the signature of the security update. If the verification is successful, the integrity of the security update is then confirmed.

Once a Revere node has verified the integrity of a security update, it will forward the security update to its children. Note that the Revere node will not make any changes to the security update at all; in particular, it will not touch the signature field. When those children receive the security update, they will apply the same procedure to verify the integrity of the security update. Normally, when a Revere node receives a security update, the update will have already passed a list of intermediate Revere nodes; but as long as the signature is verified, the integrity of the update can still be trusted.

5.2.4 Authenticity of Security Updates

A Revere node must ensure that a received security update originated from a dissemination center. Given that every Revere node uses the public key of a dissemination center to verify newly received security updates, only when the private key of that dissemination center is used to sign those security updates can the verification succeed. If the private key of the dissemination center is well protected and only the center itself knows its own private key, a Revere node can then guarantee that a verified security update has indeed originated from that dissemination center, and thus can trust the authenticity of the security update.

Clearly, the private keys of dissemination centers must be carefully protected. If, despite such care, the private key of a dissemination center is compromised, disastrous attacks can be launched; e.g., attackers now can easily impersonate the dissemination center. Revere nodes, unaware of the leakage of the dissemination center's private key, will "successfully" verify the security updates that are actually forged by the attacker, and even be fooled into taking actions based upon such updates.

Four problems must be solved. First, how can Revere quickly detect that, instead of a dissemination center, an attacker is sending (forged) security updates by impersonating a center? Second, after a dissemination center recognizes that its private key is compromised, its public key can no longer be used for security update verification; in this case, how can the dissemination center revoke or invalidate the current public key? Third, how can those Revere nodes obtain the next public key of the dissemination center? Fourth, should old security updates be resigned and redelivered? We call the first problem impersonation detection, the second problem key invalidation, the third problem key distribution, and the fourth problem security update redelivery.

5.2.4.1 Impersonation detection

This issue is a difficult problem and still under investigation, we discuss this issue here, rather than in Section 5.5 on open issues to maintain the logical connection of discussions in this chapter. In our current system, a reverse traversal mechanism is designed for this purpose.

After a Revere node has successfully verified a newly received security update, if it is still suspicious about the authenticity of the update, it can initiate an impersonation detection process as follows:

- This Revere node first reports the security update in question to all of its parents.
- Upon the receipt of this update, every parent will first verify the update; if verified, that parent reports further up to its parent. This repeats until the dissemination center receives this reported security update.

Every parent will only report one copy of such a security update to its parents; otherwise, the dissemination center can be imploded by the updates when many Revere nodes initiate a report of the same update.

- The dissemination center diagnoses the reported security update, and makes sure the update was indeed recently sent by itself. If the center has no knowledge of this security update, and some attacker has compromised the private key of the dissemination center and is impersonating the center—the center should immediately trigger the key invalidation procedure.

Each Revere node could randomly (with very low probability) test a security update this way. In this fashion, unless a forged update is injected very close to the "edges" of the Revere graph, some node will probably be triggered to initiate the impersonation detection process.

5.2.4.2 Key invalidation

Here we discuss the procedure of invalidating the public key of a dissemination center once its private key is compromised. Recall that the public key is distributed all over the RBone. Obviously, the center itself cannot distribute a new public key to replace the old one. (If a new public key to distribute is signed by the current private key of the dissemination center, the attacker who also has the private key could easily impersonate the center and distribute a forged public key.) Rather, the dissemination center sends out a key invalidation message to declare that its current public key should be invalidated.

The invalidation message is signed by the broken private key, and delivered in the same way as normal security updates. Figure 5.2 shows a key invalidation message. The

Fig 5.2 The key invalidation message

"type" field indicates this is a key invalidation message, rather than a normal security update. The "key serial no" field specifies which public key to invalidate, in case a dissemination center has a list of public keys. Note that the key invalidation message is really simple—it does not contain any extra information at all. The reason is that any extra information cannot be trusted by a Revere node at all, since the attacker who has the private key can easily fabricate or change those fields. The key invalidation message is designed to only pass a single fact to Revere nodes—a public key must be invalidated, but nothing more than that.

When a Revere node receives a key invalidation message, it will verify the message using the current public key of the dissemination center (or more accurately, the public key indicated by the "key serial no" in the key invalidation message); if verified, the current public key will be discarded, and the key invalidation message will be forwarded to other security updates. Figure 5.3 depicts such a procedure. There might be a loop when delivering a key invalidation message (just as when delivering normal security updates); however, when a Revere node receives such a message for the second time, the public key to invalidate will have already been discarded and no action will be taken (and no action needs to be taken).

The repository servers will keep all key invalidation messages. When a repository server receives a pulling request from a reconnected node for missed security updates, based on the disconnection period indicated in the pulling request, the repository server can determine whether the reconnected node may also have missed some key invalidation

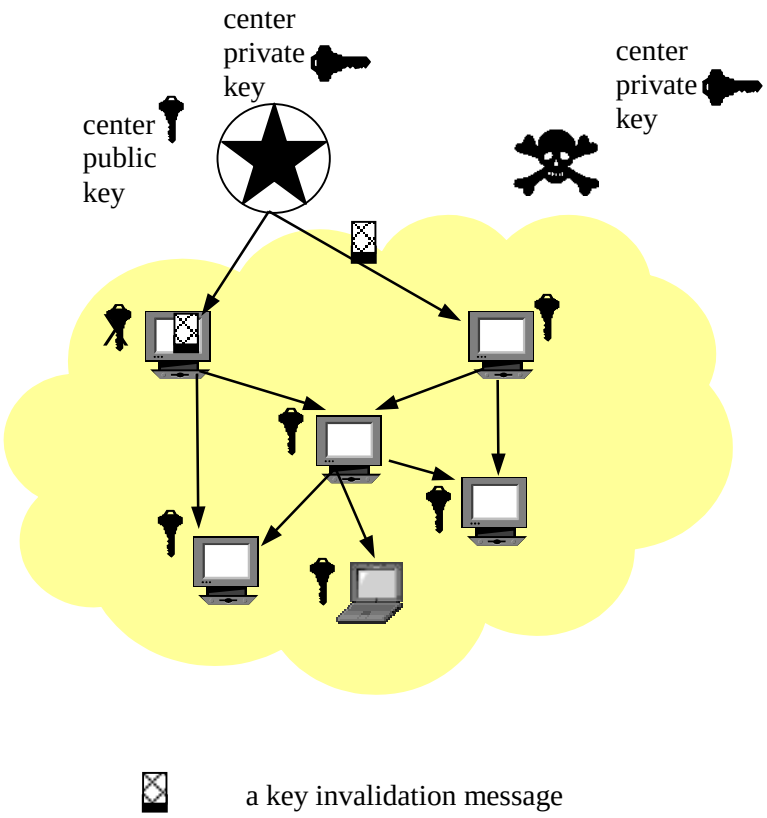


Fig 5.3 Center public key invalidation

messages. As a result, the reconnected node may also receive a key invalidation message—it will handle this message in the same way as described above.

An attacker may launch several kinds of attacks against the key invalidation mechanism. First, the attacker can try to suppress the invalidation message. However, an RBone is already a resilient network with built-in redundancy to protect normal security updates. Given that a key invalidation message is delivered in the same way as normal security updates, every Revere node can have multiple resilient paths to receive the key invalidation message. Second, the attacker who has broken the private key can create his own invalidation message, but doing so will destroy any benefit he receives from

cracking the key. This helps a Revere node invalidate the public key that it *should* invalidate.

However, the attacker does achieve one goal when impersonating a dissemination center and sending out key invalidation messages: if the dissemination center has not detected the leakage of its private key yet and sends out a security update, those Revere nodes who have invalidated their current public key will not accept the security update. Revere corrects this problem by adding reverse path forwarding of key invalidation messages, similar to what's done in impersonation detection. Doing so, the dissemination center will be able to quickly detect not only the leakage of its own private key, but also the fact that the attacker is disseminating a key invalidation message to disable the current public key of the dissemination center. The dissemination center then starts its own key invalidation procedure as described above, and begins other actions that are discussed below in the key distribution procedure.

5.2.4.3 Key distribution and key switch

After invalidating the dissemination center's old public/private key pair, a new pair must be created. Or, if a dissemination center assigns an expiration time to its current public/private key pair, a new pair must also be provided when the old key pair expires. The new public key must be distributed to all Revere nodes, possibly millions of them. Manual distribution would be daunting. We foresee two possible methods for distributing a new dissemination center public key.

- **Key distribution center.**

  Although manually distributing a new key to millions of Revere nodes is prohibitive, we can manually distribute the new key to several secondary trusted centers and let every Revere node contact those trusted centers.

Many schemes that use public-key infrastructure at high scale have this problem, and we can leverage any good solution others might create.

- **Pre-installation.**

  This method pre-installs a series of dissemination center public keys at every Revere node. After a public key is invalidated or timed out, the Revere node will then automatically switch to the next available public key in the series. One must be careful here in protecting the corresponding series of private keys of the dissemination center. If the whole series of private keys are compromised, for instance, it will be self-defeating to switch to next already-compromised private key in the series. One can imagine that separately protecting every private key in the series will be helpful.

With either key distribution method noted above, a version number could be associated with a public key, and every security update could be labeled with this version number. If a Revere node misses a key invalidation message, when it receives a security update that carries a different version number, it then knows that a different public key of the dissemination center should be used, or an adversary has modified the version number to fool recipients. In this case, the Revere node can consult with repository servers to see if it has missed some key invalidation messages. If not, it means the current public key is still effective, unless it times out; otherwise, it should switch to the next public key.

5.2.4.4 Security update redelivery

During the period between key compromise and key invalidation, the security updates received at a Revere node or a repository server, even though verified as authentic, may

be forgeries, or may indeed be valid updates. The forgeries should be purged, and the valid updates should be replaced with their new versions.

After switching to the new public key, the dissemination center will do two things. First, it will send out an "estimated corruption time" message, signed by the new private key. (Note that the estimated key corruption time is not carried via the key invalidation message at an earlier time, for such information can be fabricated by an attacker as well.) Every Revere node can then distinguish between security updates disseminated before the center's key is corrupted and those after the corruption (recall that every security update carries a timestamp that indicates when the security update departed from the center). Second, the dissemination center will resend those security updates that were sent after the estimated key corruption time but before beginning the invalidation of the center's public key. These are unfortunately signed using the corrupted private key, so those security updates are re-sent, signed using the new private key.

5.2.5 Availability of Security Updates

The availability of security updates is supported by resilient delivery of security updates over RBones. As we discussed in Chapter 3, every Revere node can choose to have multiple as-disjoint-as-possible delivery paths at its own discretion. During dissemination, a Revere node expects a copy of the same security update from every parent (under normal conditions). Unless an attacker corrupts every path, a Revere node will still be able to receive a copy of the same security update.

The availability of security updates is further strengthened by the ability to pull missed security updates from more than one repository server. If a repository server does not return a complete set of missed security updates, a Revere node can discover the missing updates by simultaneously consulting other repository servers. A Revere node

can also check with repository servers every so often, depending on the node's level of paranoia.

Multiple dissemination centers can also be set up to provide the same set of security updates, thus also improving availability.

5.2.6 Replay Prevention of Security Updates

To some degree, the replaying of security updates helps dissemination, but it can also trigger a flooding attack if not properly handled, because Revere is essentially an amplification mechanism for sending messages. One initiator of a security update can potentially have its message delivered to thousands or millions of nodes. If not handled carefully, Revere could be misused to flood the network or Revere-participating nodes. While Revere authentication mechanisms would ensure that no improper actions were taken on the basis of forged Revere messages, unless some care is taken, Revere would tend to disseminate replayed legitimate Revere messages. Thus, a Revere node should not just check authenticity of a message, but should also determine if the message has been replayed.

Replay of security updates is prevented through the "duplicate check" process at each Revere node (see Chapter 4). By keeping record of sequence numbers already seen, duplicate security updates—both those that are caused by dissemination loops and those that are replayed by attackers—will be detected and dropped, and thus not forwarded to other Revere nodes. Here, those historical sequence numbers are recorded in ranges, since the sequence numbers carried by security updates are usually continuous.

5.2.7 Secrecy of Security Updates

As an open-membership system, Revere does not address the secrecy of security updates. However, if this turns out to be a necessity, we do not exclude the possibility of

adding such support. Essentially, this will turn Revere into a closed-membership system, in which security updates in transit must be encrypted, and the key used for decrypting security updates at each Revere node must be carefully handled. Typical issues include, but are not limited to: how to distribute a key to a newly joined Revere node, how to generate a new key when an existing Revere node withdraws from Revere, and how to handle a compromised Revere node that has the secret key to decrypt security updates.

The research community that works on secure multicasting has been studying exactly the same set of problems along this line. Revere should be able to leverage those research results if the secrecy of updates becomes important.

5.2.8 Hop-by-Hop Security Update Protection

When a Revere node receives a corrupted security update, it needs to determine which parent forwarded the update so that the node can remove this parent and re-select a new parent.

Revere provides the flexibility for a parent node and a child node to strengthen the security update forwarding operation between them. When a security update is forwarded from a parent node to a child node, in addition to the signature signed by a dissemination center, the parent node can further strengthen security by also signing the security update, thus preventing parent spoofing. The same method for protecting node-to-node control messages (as described in the following Section 5.3) can be applied here.

5.2.9 Implementation

To implement the protection of a dissemination process, a security update protector is implemented at each Revere node. At each of these nodes, the security protection is essentially done by a security update protector—including security update integrity and authenticity protection, key invalidation and switch, replay prevention, etc.

Figure 5.4 shows an example of calling the security update protector function, where the function in an input job, *HandleIncomingSecurityUpdatesFromNet(),* calls the *HandleIncomingSecurityUpdatesFromNet()* (same name) function of the security update protector to verify the signature of an incoming security update.

Figure 5.5 shows another example of interacting with the security update protector at a Revere node. An output job function, *forwardSecurityUpdates(),* calls *secureSecurityUpdatesToNet()* to strengthen the security of a security update that is about to be transmitted: if this Revere node is a dissemination center, the security update will be signed using the center's private key; otherwise, hop-by-hop security update protection can be applied (as described in Section 5.2.8).

The security update protector can be regarded as a "security monitor" in terms of dissemination since it monitors and filters all inbound and outbound security updates. Every Revere node can configure its security update protector to enforce different security policies.

input job: *HandleIncomingSecurityUpdatesFromNet()*

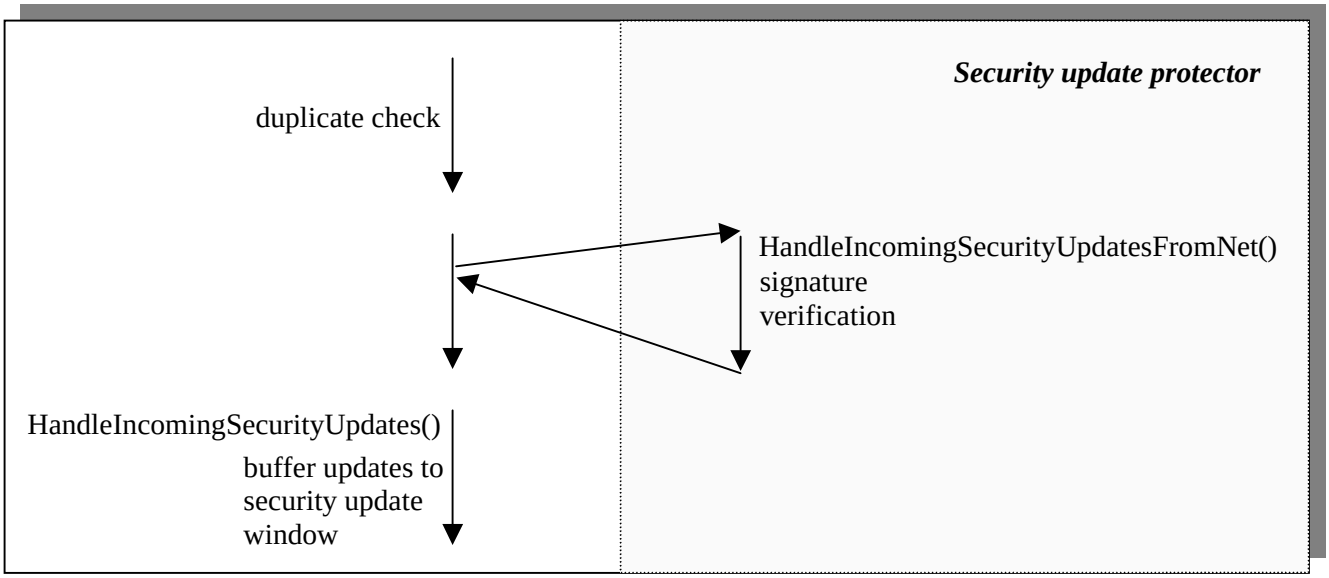


Fig 5.4 An input job function calling the security update protector

output job: *forwardSecurityUpdates()*

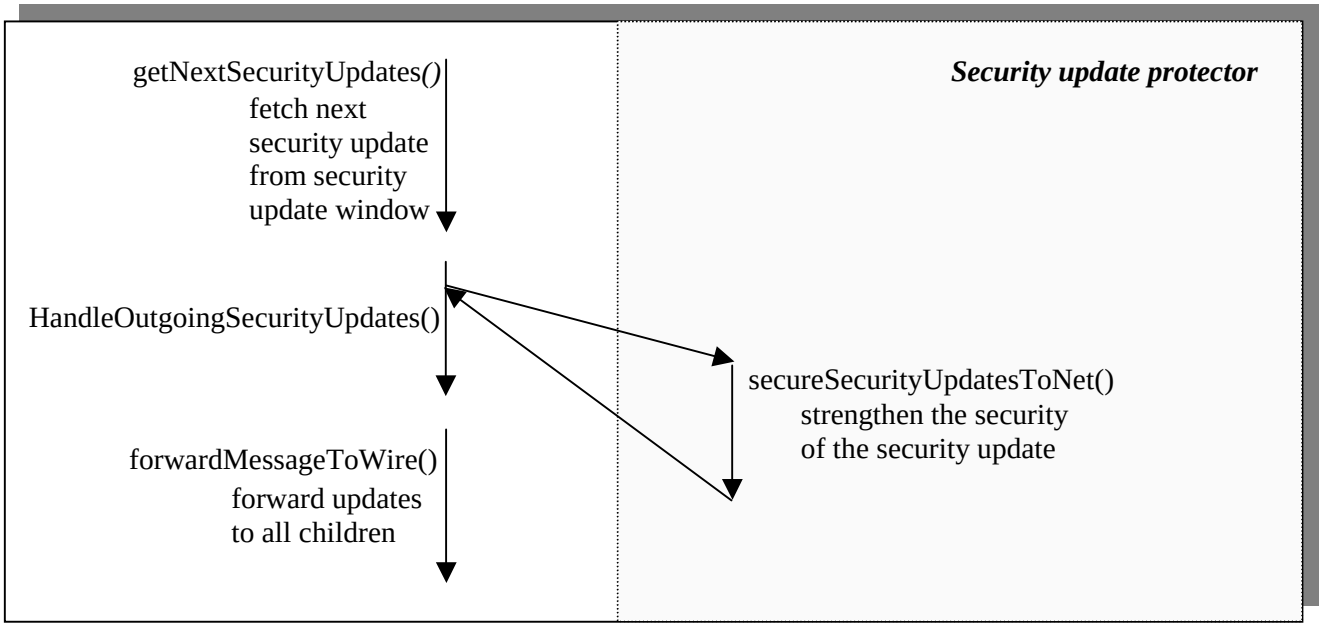


Fig 5.5 An output job function calling the security update protector

5.3 Securely Building and Maintaining RBones

5.3.1 Objectives

The security of Revere depends on building an RBone with good dissemination properties and ensuring that its structure remains sound. A Revere node needs to authenticate another Revere node if it wants to accept information from that node. For example, a new Revere node must authenticate a potential parent before accepting the potential parent as a new parent, and an existing Revere node must authenticate a potential child before allocating a spot for adopting the potential child as a new child. Moreover, Revere uses a number of control messages between parents and children, and Revere needs to secure those messages based on the needs of every involved Revere node. For example, a Revere node must verify heartbeat messages from parents or children in order to not be fooled in believing that a dead (or broken) parent (or child) is still healthy and alive, and a Revere node must verify a termination request from a parent (or a child) before tearing down the connection with that node.

To protect the integrity of an RBone, Revere authenticates nodes and secures those control messages on a peer-to-peer basis, or hop-by-hop basis. In other words, every Revere node tries to ensure that the interactions with its parents or children are secured, but not beyond that. An alternative approach is to employ path-wise security, where a Revere node checks the authenticity of every node on all security update delivery paths and verifies the information of every hop. This alternative greatly increases the overhead and complexity, whereas we believe a hop-by-hop model should serve well, because a large percentage of Revere nodes are benign, and every Revere node can still verify the digital signature of a security update during the dissemination process, if the parent of a Revere node's parent is corrupted. The goal is that, through hop-by-hop security

enforcement, there will be chains of trust starting at an RBone's dissemination center and reaching most Revere nodes (if not all).

Peer-to-peer security enforcement begins with security scheme negotiation. Since there is no ubiquitous security scheme for all nodes on a large-scale RBone, nor is one level of trust appropriate for all situations, every Revere node may implement a different set of security schemes, perhaps with different orders of preference. Different parameters may also be used, even if Revere nodes employ the same security scheme. Therefore, a Revere node needs to select a security scheme used for exchanging messages with each of its peers. If necessary, different schemes can be used for sending to and receiving from a given peer. Choosing the appropriate schemes requires a negotiation, and must be done securely. If the negotiation is successful, proper security schemes can then be imposed on messages exchanged between the nodes.

Flexibility in implementing and supporting different security schemes is also important. The code for a special security scheme should be easily plugged in, and easily unplugged when not needed. We will show in Section 5.3.3 that Revere implements each security scheme via a pluggable (and unpluggable) security box.

5.3.2 Peer-to-Peer Security Scheme Negotiation

Security scheme negotiation is triggered when a node wants to send another node a message, but finds that no security scheme has been chosen to protect this message. Figure 5.6 illustrates a security scheme negotiation procedure between nodes A and B, initiated by node A. The following is a stepwise explanation of the negotiation:

1. Node A first sends a *negotiation_start* message to B in plaintext, indicating an ordered list of A's preferred security schemes for incoming messages from B. More broadly, the message can also include specific parameters for every security scheme.

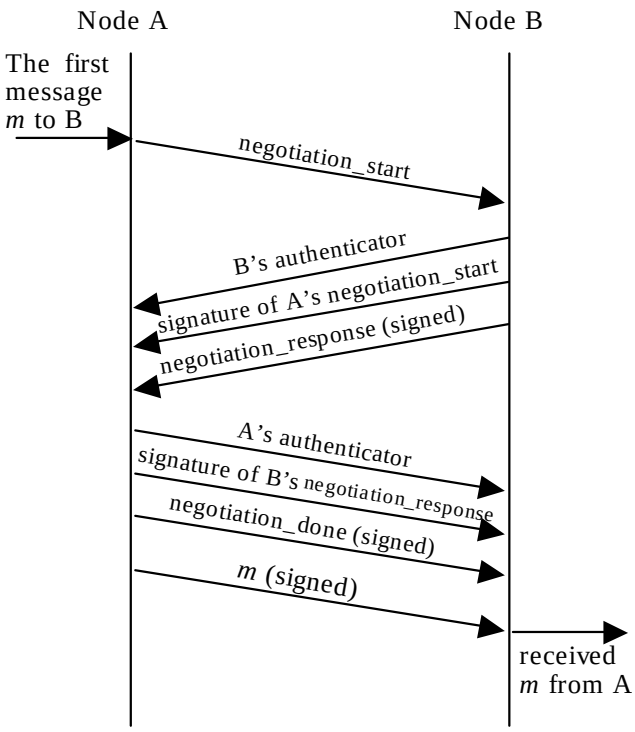


Fig 5.6 Peer-to-peer security scheme negotiation

2. Node B, upon receipt of the *negotiation_start* message, selects a scheme that B itself supports and that A also prefers, and creates an authenticator for B itself using this scheme. Node B sends to A the authenticator, a signature of A's entire *negotiation_start* message, and a signed *negotiation_response* message. The signing is done by using the scheme just selected. The *negotiation_response* message contains the scheme that B just selected and an ordered list of B's preferred security schemes for incoming messages from A. The *negotiation_response* message is signed using the scheme just selected.

3. Node A authenticates B, verifies the signature of its initial *negotiation_start* message to ensure it has not been tampered with, and verifies the *negotiation_response* message. If all are verified, A then chooses a scheme that B prefers and that A itself

also supports to protect its messages toward B. Note that this scheme is allowed to be different from the scheme that B chose for B's messages toward A. Node A sends an authenticator toward B, using the scheme A just selected. To ensure B that its *negotiation_response* was not tampered with, A sends back a signature of the message. Node A also sends a signed *negotiation_done* message toward B, indicating the scheme that A selected and ending the negotiation.

If any of these steps fail, the negotiation will fail, and no security scheme will be selected for communication between the two nodes. For example, if B cannot select a scheme successfully, B cannot, and will not, respond to A's negotiation request, and A will finally time out and give up (note there is no appropriate messages from B to be sent toward A, since such a message cannot be protected without a scheme chosen by B). If all steps succeed, the negotiation succeeds, and messages can begin to be forwarded from A to B (such as message *m* in Figure 5.6), or vice versa, protected by using the selected security schemes. When the scheme for messages from A to B is the same as that for messages from B to A, A and B are enforcing symmetric security schemes; otherwise, they are asymmetric schemes.

Figure 5.7 shows a secured version of the three-way-handshake procedure. Compared to Figure 3.4 in Chapter 3, the AttachReq message triggers a security scheme negotiation between the potential parent and the potential child (supposing the potential child has never communicated with the potential parent before). Only when the negotiation succeeds will the three-way-handshake procedure continue. Also, all those messages used in the three-way-handshake are now signed. Using the security scheme just negotiated, the AttachReq and AttachConfirm message are signed using the scheme

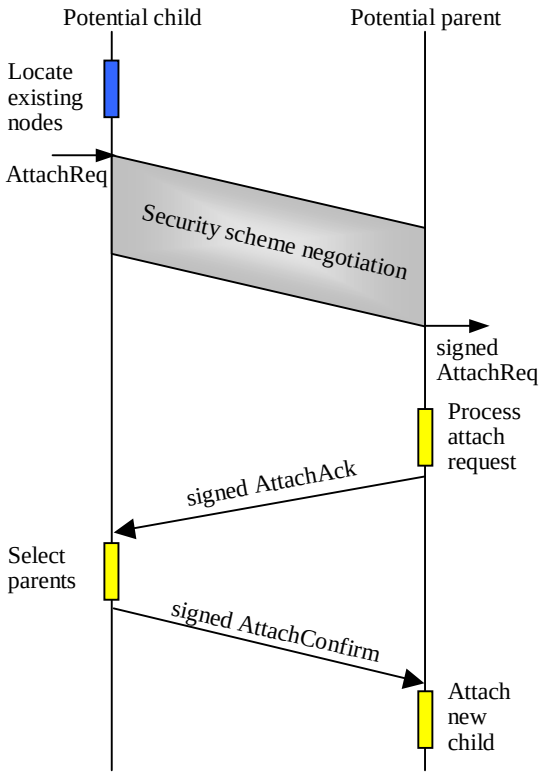


Fig 5.7 Secured three-way-handshake procedure

that the potential parent prefers, and the AttachAck message is signed using the scheme that the potential child prefers—all must be verified prior to use.

5.3.3 Pluggable Security Box

Revere implements an extensible architecture to support various security schemes. As in [Li *et al.* 2002], each security scheme can be added by plugging in a corresponding security box. This architecture allows a Revere node to choose a specific security scheme based on the desired level of protection.

A security box can be viewed as a security monitor that is responsible for node authentication. It protects RBone activities such as the join procedure or RBone maintenance. A security box allows a node to authenticate other nodes or authenticate itself to another node. A security box only allows trustworthy RBone activities.

A security box can also be viewed as a message filter (Figure 5.8). All control messages sent and received must pass through the security box. Incoming messages are accepted or rejected based on trust and authenticity. Outgoing messages are inspected and stamped with authentication information. Every RBone control message, including heartbeat messages and those used during the join procedure, is signed by its sender's security box and verified by its receiver's security box.

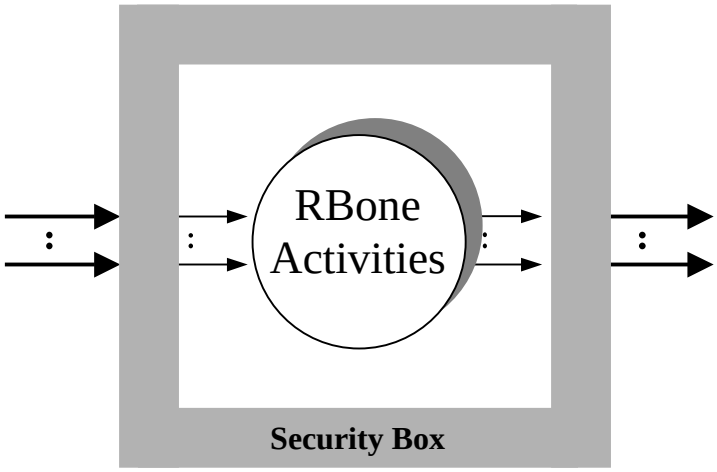


Fig 5.8 Security box

Note that security updates do not pass through any security boxes; instead, they are protected through the security update protector. In particular, every Revere node is required to enforce the same algorithm to verify the signature signed by a dissemination center.

Many security box implementations are possible, each providing a different level of node authentication, message verification, replay prevention, and possibly secrecy. The level of protection provided depends entirely on the particular security box implementation. Moreover, when providing different levels of protection, different

security schemes require different amounts of infrastructure (which may or may not be available), and have different levels of overhead.

A simplest security scheme is the *null* scheme, which actually does not provide any protection and can only be used when a Revere node does not require security. Having such a scheme can help demonstrate the added cost of a security architecture.

More complex security schemes can be supported. For instance, symmetric-cryptography-based security schemes can be implemented via a security box using Kerberos or other key distribution mechanisms, and asymmetric-cryptography-based security schemes can be implemented via a security box that relies on a public key infrastructure. We discuss these two exemplary security schemes in the following two subsections.

5.3.4 Security Box Example 1: Using Kerberos

5.3.4.1 The Kerberos model

In order to authenticate to another node *S* using Kerberos, a node *C* obtains a ticket and then presents that ticket to *S* for authentication. At an extremely high level, *C* sends a request to Kerberos to authenticate to *S*. *C* receives (in the end) a session key for talking with *S*, encrypted with a key it shares with Kerberos, along with a ticket that it can send to *S*. The ticket contains (among other things) the identity of *C* and *S* and a session key for talking with *C* encrypted with a secret that *S* shares with Kerberos.

To authenticate to *S*, *C* sends a Kerberos authenticator (a time-stamp, a checksum, etc.) encrypted using the session key to *S* along with the ticket. *S* can obtain the session key using the key it shares with Kerberos and use it to decrypt the authenticator and therefore verify the authenticity of *C*. *S* can (optionally) send an authenticator back to *C*, again encrypted with the session key, allowing *C* to authenticate *S*.

Once established, the session key used for authentication of the session can now be used by *C* and *S* for various security purposes.

5.3.4.2 Integration with Revere

Integrating Kerberos into Revere is straightforward. When a first Revere node needs to authenticate itself to a second Revere node using Kerberos, both nodes must share a secret with Kerberos. The first Revere node then can request from Kerberos (1) a session key between the two nodes, which is encrypted such that only the first Revere node can decrypt, and (2) a ticket to send to the second node, which is the only node able to decrypt the ticket to retrieve the session key. Doing so, a session key will be distributed to both nodes. This session key can then be used for both authentication and message protection between the two nodes.

Since Revere runs at Internet scale, the two communicating nodes may be located at two different domains, each with a different Kerberos server. The Kerberos infrastructure is already designed to handle this case. While a Revere node may have to communicate with several Kerberos servers, it will eventually end up with a session key and ticket that have been generated by the Kerberos server associated with the second Revere node.

5.3.5 Security Box Example 2: Using Certificate Authority Hierarchy

One security box that we have constructed is based on a hierarchical infrastructure of public key certificate authorities (CA), where recursively the CA at one level (the parent) produces certificates for the next level down (the child). The public key for the CA at the root of the hierarchy, the highest level, is universally known.

With this scheme, the verification of a node's public key, which is equivalently the authentication of the node itself, is straightforward. A node *n* can contact its associated

CA, *CA(n)*, to obtain a chain of certificates. On this chain, the first certificate certifies the public key of node *n* by *CA(n)*, the second certifies *CA(n)*'s public key by *CA(n)*'s parent, *CA(n-1)*, on the certificate hierarchy, and so on. The last certificate on the chain is signed by the root. By verifying such a chain of certificates, the node's public key can then be authenticated.

Note that the set of certificates needed to certify a node's public key is static in this scheme. A node can therefore cache all of the certificates it will need to authenticate itself to any other node.

Because a Revere node's public key can be verified, the messages from this node can also be protected. When this node needs to send messages to other Revere nodes, it can sign every message using its private key. The digital signature of any message from this node can be verified by other Revere nodes using this node's public key.

5.3.6 Summary: Generality and Particularity

Because of the Internet scale that Revere addresses, Revere nodes are inevitably heterogeneous. It is unrealistic to assume that they enforce a uniform security scheme with the same parameters and policies. Nonetheless, these Revere nodes must still be able to communicate, and communicate securely. This requires Revere to be designed in a sufficiently general way to accommodate such heterogeneity. On the other hand, the particularity of every Revere node calls for the capability of easily tailoring security enforcement to local needs.

The peer-to-peer security scheme negotiation allows two arbitrary Revere nodes to check their compatibility before cooperating on RBone operations, addressing the need to be general. The implementation of pluggable security boxes supports extensible security

architecture and allows special security enforcements to be easily added or removed, addressing the need to be specific.

Moreover, such generality applies to other distributed systems that have similar security requirements. Many Internet applications need to address the problem of enabling secure communication between machines that are distributed among different administrative domains with different security policies and schemes. To do so, they need capabilities similar to that provided by Revere.

5.4 Attacks and Countermeasures

Attackers can attempt to break into an RBone or rely on other resources to subvert Revere. This section describes possible attacks and the countermeasures employed by Revere. We divide the attacks into two categories: (1) attacks on the dissemination procedure, and (2) attacks on RBone formation and maintenance.

5.4.1 Attacks on Dissemination Procedure

5.4.1.1 Suppressing, misdirecting, or tampering with security updates

Upon the receipt of a new security update, a compromised parent node on an RBone may drop or misdirect the security update, instead of forwarding it to its children. To address this, Revere adopts a redundancy mechanism: every node can choose to have multiple parents forward security updates. As discussed in Chapter 3, Revere tries to ensure that a node chooses parents with disjoint dissemination paths, thus reducing the impact of compromised nodes.

A compromised parent may also tamper with security updates. Although a node can ascertain that a received security update has been corrupted by verifying the signature carried in the update, and thus not be deceived into accepting the update, the node will still miss an authentic copy of the security update. Redundancy is again the solution.

The difference here is that each node must be able to verify whether a received security update is authentic. As discussed in Section 5.2, every node can use the public key of the dissemination center to verify the signature of every received security update.

Moreover, with either case above, if a node did not receive an authentic copy of a security update from one parent (but it did from other parents), the node will regard this parent as problematic and remove it, and begin to search for a new one.

Note that each Revere node can specify its own policy for fending off the attacks described above; for example, it can specify the redundancy level of the in-bound paths. Through built-in redundancy, we hope there can be at least one path bringing timely authentic security updates to a Revere node, even when a varying number of Revere nodes have been subverted. (If every Revere node chooses only one parent, an RBone will become a tree rooted at a dissemination center.)

5.4.1.2 Replaying security updates

As discussed earlier in Section 5.2.6, replay attacks must be prevented, and Revere's duplicate checking capability achieves this goal by dropping replayed security updates.

5.4.1.3 Compromising the private key of a dissemination center

If the private key of a dissemination center is compromised, it is disastrous in that an attacker can now disseminate malicious information under the mask of authentic security updates. Strong cryptography should be used, and the key of a dissemination center must be protected very well. In the event that the key does become compromised, we have proposed methods to detect the impersonation, to invalidate the compromised key and then distribute a new key (see Section 5.2.4).

5.4.1.4 Breaking into a repository server

A repository server keeps old security updates. Two undesirable things can happen if a node tries to pull missed security updates from a compromised repository server: the node receives a tampered security update, or the node receives an incomplete list of missed security updates. In the former case, since all security updates carry the signature of the dissemination center, pulling nodes will detect the tampering when trying to verify the signature. In the latter case, a node pulls updates from one repository first, then verifies completeness by consulting one or more other repositories. The only information needed is the sequence numbers of the missed security updates. For example, after retrieving missed updates with sequence numbers 10 to 15 from the first repository, if the second repository reports that it has security updates with sequence numbers up to 17, the node can then try to pull security update 16 and 17 from the second repository. In case that the second repository server was lying, the node will detect this by verifying the authenticity of security updates.

5.4.2 Attacks on RBone Formation and Maintenance

5.4.2.1 Attacking security scheme negotiation

During security scheme negotiation, a compromised Revere node may try to trick the other side (the benign side) into using a weaker scheme to verify messages from the compromised node. Revere prevents this problem. As shown in Figure 5.6, whether the compromised node is the initiator or not, it must use one of the schemes already specified by the other side to authenticate itself and sign its response.

5.4.2.2 Impersonating another node

An attacker can try to impersonate another Revere node and send forged messages for joining or maintaining an RBone. Revere allows each node to use its preferred security

box to check incoming messages. For example, when using the security box described in Section 5.3.5, every message is signed by the sender, whose public key can be authenticated by querying the certificate hierarchy. The recipient can thereby ensure that the message is indeed from the sender as labelled in the message.

5.4.2.3 Replaying RBone messages

An attacker can also try to replay previous RBone control messages to fool a Revere node. As a standard technique to prevent this, a sender can include a random number preset by the recipient inside the message, and sign the whole message. As shown in Figure 5.9, if attacker *a* eavesdrops on a message from *y* to *x* and replays it, *x* can detect that the message, which carries the same random number *r* as a previously received one, is a replayed message.

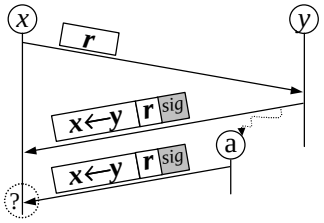


Fig 5.9 Replay attack prevention using a random number

5.4.3 Compromising Security Update Secrecy

In general, Revere does not handle this problem. Its free subscription paradigm implies that disseminated security updates are open to the public. For instance, when disseminating the signature of a newly discovered virus, one would like to see as many machines as possible obtain the security updates regarding the new virus.

On the other hand, we can treat this as a secondary goal. With this in mind, preventing the leakage of a security update is challenging due to the scalability constraints of Revere. If a different key must be used for sending security updates to

each Revere node from a dissemination source, the dissemination source has to keep all the public keys of each Revere node; this is not scalable. If a shared key is used, then joining and leaving the Revere system must be handled in a highly secure fashion, so that each new member can be trusted, and each previous member cannot re-gain the security updates disseminated after it left Revere.

5.5 Open Issues

5.5.1 Securely Monitoring Dissemination Progress

One open question is how Revere securely monitors the progress of dissemination. So far there is no feedback mechanism designed in Revere, partly because of the scalability concerns when handling feedback from millions of Revere nodes, but mostly because of the difficulty in securing a feedback mechanism if there was one.

In the following paragraphs we look at two different conceivable feedback mechanisms: random feedback sampling and aggregated feedback collecting. We will see that both address the scalability issue well, but still face serious security challenges.

The random feedback sampling method randomly chooses a number of Revere nodes, and checks the set of received security updates for each of them. Given that the center has no record of all current Revere nodes in an RBone (except its own directly connected children), the sampling method has to rely on every Revere node to proactively and randomly elect itself (supposing every node uses a probability, say 0.01), and then report to the center. The center, based on the sets of received security updates from all reporting Revere nodes, deduces the progress of dissemination. Unfortunately, the dissemination center has no idea whether a reporting node is authentic or not. If every corrupted node in an RBone reports to the center that it has received a complete set of security updates, while only a very small percentage of uncorrupted nodes report their status, the center

can be fooled into believing that the dissemination has been going well. The center might require every report to be signed (using public key cryptography for example); even if the sender authenticity of a feedback message is trusted, this still does not necessarily guarantee that the content of the message is trustworthy.

The aggregated feedback collecting method requires every Revere node to provide feedback regarding a specific security update. Every Revere node waits for reports from all of its children (with a timer set), aggregates the reports into one report, and sends to every parent. The report contains a list of reached descendents (those that reported receiving the security update), and a list of unreached descendents (those that did not report before timing out). Through the aggregation of multiple reports into one report, those repeated entries will be removed, and a node that appeared in both lists will be removed from the list of unreached nodes. With luck, the dissemination center will finally receive a report it can use to check the progress of dissemination. This method also faces security issues. A corrupted Revere node may report all its descendents as having received the security update, even if it has not forwarded an authentic copy to any of them. As a result, if a Revere node happens to be isolated by corrupted Revere nodes, the fact that it has not received the security update will not be reported to the dissemination center at all.

5.5.2 RBone Intrusion Detection and Reaction

As discussed earlier, Revere has been designed to be robust against various attacks as they occur. On the other hand, it is still a difficult problem for Revere to globally monitor the health of an RBone. In other words, a distributed intrusion detection system for Revere is an open question. The open-membership principle of Revere makes this more challenging—a compromised node can easily join an RBone as long as it can attach

itself somewhere. Although Revere has been designed to be resilient to interruption threats and other attacks, it would still be very useful if Revere could identify those compromised nodes as early as possible, instead of waiting until attacks are launched. Moreover, after a Revere node is identified as unreliable by other Revere nodes (for example, a Revere node may detect that one of its parents does not cooperate in forwarding security updates), how can the corruption of the node be securely and reliably reported, and how can all the other Revere nodes be alerted?

5.5.3 Denial-of-Service Attack

Denial-of-service (DOS) attacks are always particularly challenging. Similar to DOS attacks launched on web servers [CERT 2000], attackers can attempt to flood a dissemination center to stop its normal dissemination operation. Revere could also face another type of DOS attack, one caused by infinite join. An attacker could try to corrupt a certain number of nodes first, and then let those nodes continuously try to become children of every uncorrupted node through the join procedure. If successful, all those benign nodes may never have enough space to accept new children; new nodes will become children of the corrupted nodes and will be unable to receive authentic security updates. Studies addressed in Section 5.5.2 may prove useful for solving this problem.

5.6 Conclusions

The security of a distributed system has always been challenging. Revere can be regarded as a distributed system that consists of Revere nodes over the entire Internet. The large scale, the node heterogeneity, and the various forms of attacks all pose challenges to securing Revere. Although there are still open issues, such as securely monitoring dissemination progress, RBone intrusion detection and reaction, and denial-of-service attack prevention, the security of Revere is fully addressed in this chapter.

The security of Revere is divided into two aspects: securing the dissemination process, and securing the RBone structure. As discussed in this chapter, the security of dissemination process concerns the integrity, authenticity and availability of security updates. Attacks to counter along this line include suppressing, misdirecting, replaying, or tampering with security updates, breaking into repository servers, or even corrupting a dissemination center. The signing of security updates using public key cryptography is used to address this, combined with mechanisms for detecting center impersonation, invalidating corrupted keys, switching to new keys, and so on. Also, the design of Revere already helps prevent the replay attack (by checking duplicate security updates during dissemination) and interruption threats (by building redundancy into RBones).

Securing the RBone structure is challenging, as an RBone consists of large-scale heterogeneous nodes, each enforcing different security schemes. A peer-to-peer security scheme negotiation protocol was proposed in this chapter, allowing two arbitrary nodes to securely communicate with each other. Moreover, the security box corresponding to a specific security scheme can also be easily plugged in or unplugged, as exemplified in the Kerberos-based security box and certificate authority hierarchy-based security box.

CHAPTER 6

Real Measurement Under Virtual Topology

Revere provides a service for disseminating security updates at Internet scale. To understand how effective Revere is in providing this service, the characteristics of the dissemination must be evaluated. This is a critical step before widely deploying Revere over the Internet.

Among various parameters, the speed of dissemination describes the basic behavior of Revere and must be measured. Furthermore, we must measure the quality of dissemination in order to understand the resiliency of the Revere infrastructure; Revere performance in the face of broken nodes is particular interesting.

In addition to the dissemination of security updates, another major Revere activity is RBone formation and maintenance. As discussed in Chapter 3, an RBone is gradually formed by a series of join procedures, which are also employed when a node needs to adjust its position during RBone maintenance. As a result, performance data regarding the join procedure is also key to the assessment of Revere.

The difficulty arises because Revere is designed for Internet-scale deployment. Realistic measurement of large-scale distributed systems poses unique challenges. Empirical measurements can capture the true behavior of a real system, but this approach is only feasible when the system is small in scale. Simulation is more scalable, but without running real software, it is difficult for simulation tools to capture all realistic effects. We adopted an “overloading” approach to address this difficulty when measuring Revere.

In this chapter, we will first discuss what metrics to use to evaluate Revere (Section 6.1); then we will introduce the "overloading" approach used for measuring Revere (Section 6.2), which we believe also applies to other distributed systems. Section 6.3 is about the procedure of the measurement, where both measurement configurations and measurement steps will be described. Results and their analysis are covered in Section 6.4, where we discuss the results of the join procedure, dissemination speed, and dissemination resiliency, etc. Section 6.5 is on open issues, where we discuss those open problems related to the overloading approach, performance for larger-scale RBones, real-world challenges, and measurement of an RBone's physical-layer property. We conclude the chapter in Section 6.6.

6.1 Metrics

The following metrics are important for evaluating Revere:

1. ***Dissemination bandwidth*. The bandwidth spent to disseminate security updates.**

   The dissemination bandwidth, under normal conditions, is easy to evaluate. In a single round of dissemination, the inbound dissemination bandwidth per Revere node is the size of the security update multiplied by the number of parents, and the outbound dissemination bandwidth per Revere node is the size of the security update multiplied by the number of children.

   Under abnormal conditions, the dissemination bandwidth cost can be arbitrary. For instance, a subverted Revere node might try to flood its children with replayed security updates and thereby use up all the bandwidth available.

2. ***Maintenance bandwidth*. The bandwidth spent to maintain an RBone.**

   The maintenance bandwidth, under normal conditions, is also easy to evaluate. The RBone maintenance bandwidth per Revere node is mainly the size of heartbeat messages during each heartbeat interval.

   Similarly, under abnormal conditions, a subverted Revere node may arbitrarily initiate Revere messages related to maintenance.

3. ***Join latency*. The time that a new node spends before fully becoming a Revere participant.**

   More accurately, this is the time that a node spends to find the needed number of *satisfactory* parents. Note that even if the node has already attached to a certain number of parents as required, the join procedure may still not be done—some parents may have to be replaced with better quality parents (or not replaced if better parents cannot be located after certain runs).

4. ***Join bandwidth*. The bandwidth spent to join Revere.**

   Given that the messages of an RBone, including those related to join procedure, are all internal to the RBone itself, the total amount of outbound join bandwidth is the same as that of inbound join bandwidth.

5. ***Dissemination latency*. The latency for a security update to reach an individual Revere node.**

   Both average and maximum latencies in reaching a node should be assessed. Also relevant is the time needed to reach a certain percentage of all Revere nodes, including the special case of reaching *all* Revere nodes (under normal conditions).

6. ***RBone resiliency*. The percentage of working Revere nodes that still receive security updates, given that a certain number of nodes are broken.**

Because the first two metrics (dissemination bandwidth and maintenance bandwidth) are easy to evaluate, we will focus on the measurement of the remaining four metrics in this chapter.

6.2 Overloading Approach To Measuring Large-Scale Distributed Systems

6.2.1 Introduction

Conventional methods for measuring the performance of a distributed system face a scalability vs. realism dilemma. Realistic measurements of large-scale distributed systems are particularly challenging. While empirical measurements can capture the true behavior of a real system, the cost of gaining access to, configuring, maintaining, and obtaining results from more than a few hundred nodes is typically prohibitive. Simulation is a more scalable approach, but it is difficult for a simulation to capture all aspects of a real system, such as hidden costs and subtle timing effects. In addition, the simulated version of a software system is typically different from the software that would actually be deployed. The fact that simulation is usually expensive to develop but slow to run also makes a simulation-based measurement approach less favorable. In addition, simulation results must be validated against real systems.

We explore a different approach to measuring large-scale distributed systems in this chapter—the "overloading" approach. It supports multiple instances of a software system executing on the same physical node. In this approach, each individual node in a

distributed system runs the real code, and a fairly large number of nodes may be deployed on a physical node.

In the purely real world, a physical machine typically maps to one individual node of a distributed system. Via this overloading technique, however, a physical machine can be overloaded with many nodes of a distributed system, where each logical node still runs the real code and communicates with other logical nodes, just as it would in the real world. Large scale can then be achieved using multiple physical machines, each supporting many logical nodes.

In addition to providing higher-scale measurement, the overloading approach is also advantageous in that many metrics will not be affected, even if a physical node is fully overloaded. For example, storage cost and bandwidth cost will be the same no matter how many individual nodes of a distributed system are running on a single physical node. However, one fundamental issue arises—how to run a distributed system with this overloading technique while still achieving accurate measurement results. In particular, messages between the nodes will now follow different transmission paths than would be taken in the purely real world. For example, two logical nodes that are collocated on the same physical node will now communicate without crossing a wire. Also, by running multiple logical nodes on top of a physical node, resource competition between these logical nodes can slow down the processing time of various tasks, leading to inaccurate latency data that is higher than it should be.

We address these and other issues that arise from overloading in the following sections. Section 6.2.2 describes how a large-scale distributed system can run with a limited number of physical machines, using a virtual topology to assign logical nodes to a smaller number of physical machines and to model the communication between those nodes. In Section 6.2.3 we discuss measurement using this overloading approach,

including techniques that compensate for resource sharing between logical nodes on physical nodes.

6.2.2 Running Atop a Virtual Topology

The nodes of any distributed system must exist on top of some topology. When overloaded on top of physical machines, however, the nodes of a distributed system will have a different topology than they would in the real world. Such a topology, which may consist of a single machine, will not, by itself, reflect the characteristics of the topology of the distributed system.

A virtual topology can be employed to solve this problem. Each node of a particular distributed system can be viewed as attached to a particular location in a virtual topology. Such a node communicates through this virtual topology to another node, which is attached to the same virtual topology. A virtual topology can be generated using one of many existing topology generation tools, such as GT-ITM [Calvert *et al.* 1997], Tiers [Doar 1996], Inet [Jin *et al.* 2000], or Brite [Medina *et al.* 2000], depending on the characteristics of the distributed system.

With the notion of a virtual topology, a distributed system can be created as follows. After generating a virtual topology, treat each logical node in the virtual topology as an individual node of the distributed system. For each virtual node, run the software of the distributed system on top of a physical machine, where multiple instances of the software program may be invoked on the same machine. As a result, the performance of this distributed system can be measured.

While it may be possible to map all nodes of a virtual topology to a single physical machine, multiple machines will typically be required for larger scalability, each assigned a subset of nodes from the common virtual topology. The node assignment can be

fulfilled by contacting a virtual topology server that keeps track of which nodes are already assigned and which are still outstanding.

Figure 6.1 shows a virtual topology in which a distributed application runs with 20 nodes that communicate across transit-domain routers and stub-domain routers. As shown in the figure, these 20 nodes are assigned to three physical machines.

It is important to ensure that the real software still functions in this new mode of execution. One side effect of overloading is the identification of each node in the distributed system. In a real system, the address of the underlying physical machine can be used to identify a logical node. Here, since each physical node is overloaded with multiple logical nodes, logical nodes can no longer be identified using the machine address. To solve this, each node now has to be identified using the machine address coupled with some unique number, such as a TCP port number bound to the logical node

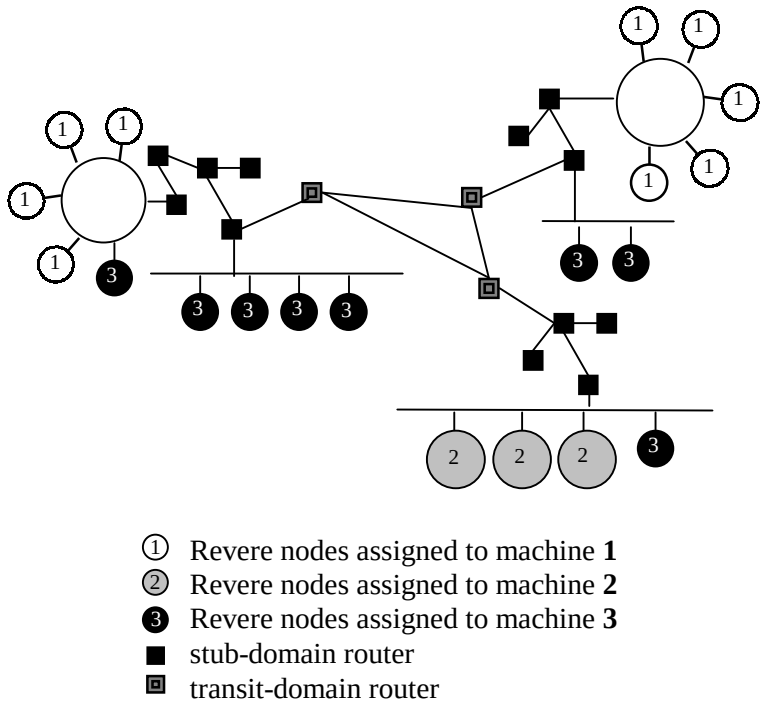


Fig 6.1 A virtual topology with 20 Revere nodes

(which is unique since two logical nodes will not be allowed to use the same port number).

6.2.3 Measuring Atop a Virtual Topology

Since overloading typically maps several logical nodes onto a single physical node, the logical nodes must share the resources of the physical node. This resource sharing can, but does not necessarily, affect the performance of individual logical nodes.

Many results obtained in a virtual topology will not differ from those obtained while running atop a real topology with the same structure. For example, whether the underlying topology is real or virtual, the storage cost or bandwidth cost incurred at an individual node of a distributed system will not typically be affected. As another example, the hop count of traveling from one node to another in a distributed system will not be affected either, when running on top of a virtual topology instead of a real one.

Also, the characteristics of the communication paths between any two nodes of a distributed system can be easily determined based on the specification of a virtual topology. For instance, if the length of every link in a virtual topology is known, the shortest path between any two nodes on the virtual topology can be calculated using Dijkstra's algorithm [Dijkstra 1959], instead of being measured.

However, logical nodes on the same physical node must share the processor and memory. Thus, the processing time of each individual node performing a particular task will be affected. Due to the overloading of the underlying physical machine, multiple nodes, if running concurrently, will cause resource contention and result in longer processing times.

This problem can be solved in three ways. The first approach is to remove the resource contention, thus causing the measured processing time on an overloaded node to

be the same as the real value. If only a single logical node at a time is allowed to proceed with full usage of system resources, the time spent by this node on a task should incur approximately the same amount of time as it would in the real world. However, this approach may require a logical node to wait for access to the resources to perform a particular task. If latency is important, this approach will not be appropriate.

The second approach is to calculate a slowdown factor and apply that to the measured processing latency. A slowdown factor can be estimated by overloading with a different number of nodes on a physical machine and comparing the impacts. For example, if a task consumes t_0 seconds when n nodes of a distributed system are evenly loaded into n physical machines, but t seconds if all n nodes are overloaded on one physical machine, we then can obtain a slowdown factor t/t_0 for physical nodes overloaded by a factor of n. This method works well when the processing time slows down linearly; otherwise, it must be carefully applied. To gain a more accurate understanding of the slowdown factor of a distributed system, measurement of overloading factors should be performed.

The third approach, using a divide-and-conquer method, is to divide the task being measured into several disjoint subtasks that are more easily measured. Here, several conditions must be met: 1) every subtask must be independent of the others, 2) subtasks must not overlap in terms of processing latency, and 3) the sum of all subtasks must be the total processing latency. For example, to evaluate the delay of forwarding a packet from source to destination, literally measuring the interval from sending time to receiving time is inaccurate when machines are overloaded. On the other hand, by dividing the whole delay into transmission delay along the wire, processing delay at each router, and queuing delay at each router, each component can be measured separately. These subtasks are usually measured in a non-overloading environment, then applied to the full system.

The first approach usually requires a new resource-control mechanism to coordinate the usage of system resources. Thus, this approach will be easier to implement for some distributed systems than it is for others. The third approach is preferable to the second if a task can be easily divided into several subtasks, and each subtask can be easily and accurately measured. It may also be possible to combine these approaches. For example, a subtask may be measured by applying a corresponding slowdown factor. In the next section, we will apply the first and the third approach in measuring Revere.

6.3 Measurement Procedure

Rather than first deploying Revere into the Internet, Revere's performance was measured using the overloading technique described above. In this section we describe our measurement procedure and justify our measurement method.

6.3.1 Configurations

To overload different numbers of Revere nodes onto physical nodes, we used a testbed that consists of ten machines. Every machine was equipped with an AMD Thunderbird 1.333 GHz CPU, 1.5GB SDRAM, and a 100 Mbps Ethernet interface.

Every virtual topology of Revere nodes was created as follows. We first used GT-ITM [Calvert *et al.* 1997] to generate a router-level topology, then attached certain numbers of Revere nodes (hosts) to each stub-domain router on that topology, and finally had a topology server assign every testbed machine the same number of Revere nodes. Transit-stub routers themselves are not Revere nodes; they are merely used in communication latency evaluation (as discussed later).

Throughout all measurements, the following configurations were used: 1) every Revere node must have **two** parents and no more than **ten** children (except that the dissemination center can have up to 30 children), 2) UDP is used for security update

forwarding from parent to child, and 3) both security updates and control messages between Revere nodes are protected using RSA-based public key cryptography with a three-level certificate server hierarchy; in particular, the signing algorithm is SHA-1/RSA/PKCS#1.

6.3.2 Phase-Based Measurement

We artificially divided the lifetime of Revere into three phases: the *join* phase, the *dissemination* phase, and the *resiliency test* phase. In real use, these three phases would overlap, but measuring them separately captures most costs appropriately. During the join phase, nodes sequentially join Revere and gradually form an RBone. After all nodes have joined, the system advances into the dissemination phase, during which the dissemination center sends security updates through the RBone to individual nodes for ten rounds. Finally, in the resiliency test phase, dissemination is tested in the face of broken nodes. We will measure join performance, dissemination latency, and dissemination resiliency in their respective phases.

6.3.2.1 Join phase

During the join phase, join latency will be artificially increased if every physical machine is overloaded with several Revere nodes, but join bandwidth should be unaffected. We evaluated the join performance for one particular scenario where all the nodes join Revere sequentially. Corresponding to the first approach discussed in Section 6.2.3, we applied a token-controlled mechanism by which a Revere node can only begin running after it is granted a token by a token server, and it must return the token after it joins Revere. By enforcing only one token for all Revere nodes on all physical machines, only one node will be in the process of the join procedure at any time during the joining phase. Other nodes may be temporarily activated when requested to interact with the

joining node. The measured results of join latency and join bandwidth should be approximately the same as the real cost of a single node joining. (In a real Revere system, there will very likely be simultaneous joins, potentially causing real contention. Measuring the join cost in this situation is under investigation.)

When a new node tries to join Revere, it will first try a three-way-handshake procedure with local Revere nodes. If there are no Revere nodes locally, or if the three-way-handshake with those nodes fails, the node then contacts the dissemination center by also running the three-way-handshake procedure. If this fails again, the center will recommend one of its children as a new contact for the new node, and the new node will start another round of the three-way-handshake procedure. This procedure repeats recursively until the new node finds two satisfactory parents.

6.3.2.2 Dissemination phase

During the dissemination phase, each node behaves in a store-and-forward manner. However, because many Revere nodes are running on a physical machine, simply measuring the interval between sending a security update and receiving it cannot reflect the true dissemination latency. Given the artificially heavy load on the physical machine, both the processing delay and the kernel-space-crossing delay will be lengthened. We solved this problem using the divide-and-conquer method described in Section 6.2.3. In this phase, we assumed no node or link failure, no malicious subversion efforts, and no any other abnormal conditions. Clearly, such assumptions are only true in a measurement environment. In the following we describe the three steps used to evaluate dissemination latency:

Step 1: Divide the security update dissemination latency

In this step, the latency of disseminating a security update is divided into three parts: the security update processing delay at every hop (including possible queuing delay), the kernel-space-crossing delay at every hop, and the transmission delay of crossing the virtual topology (Figure 6.2). Such a division satisfies the conditions of using the divide-and-conquer method as discussed in Section 6.2.3. Processing latency, communication latency and kernel-space-crossing latency are three independent non-overlapping parts, and their combination fully covers the duration of a security update dissemination.

Note that a hop here, instead of being a router-to-router hop, is a hop from one Revere node to another.

Step 2: Evaluate every individual part of dissemination latency

In this step, we evaluated each part separately. The true processing delay per hop can be measured in a separate experiment without overloading a physical node. In the same manner, the kernel-space-crossing delay per hop can also be measured. And the communication latency incurred in every hop can be calculated using the Dijkstra

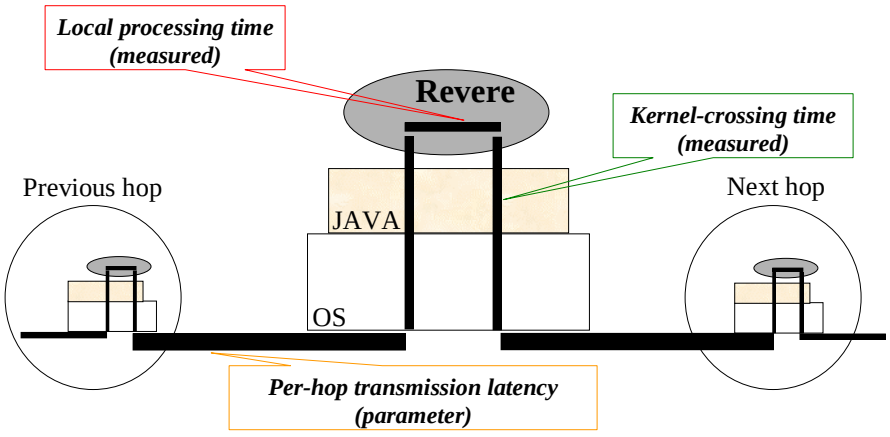


Fig 6.2 Composition of security update dissemination latency

algorithm over the virtual topology graph underneath (assuming no congestion delay on links).

It is necessary to measure both processing delay and kernel-space-crossing delay in a clean, non-overloaded environment. Figure 6.3 shows that when a testbed machine is heavily overloaded, every logical Revere node will incur prolonged local processing time and kernel-space-crossing time; neither will be accurate enough to reflect a realistic value. We found that the processing delay at a Revere node will vary with the order of forwarding security updates (we will report the measured data in Section 6.4.2.2).

Step 3: Measure hop counts (and other information) in full systems and sum

In this step, we added all dissemination latency components. Note that with a given RBone structure, the hops that a security update travels to reach a node are invariant, no matter how many nodes are simultaneously running on the same physical node. By summing up the processing delay at every hop and kernel-space-crossing delay at every hop, and adding the communication latency, we can obtain a very good approximation of the dissemination latency in large-scale scenarios.

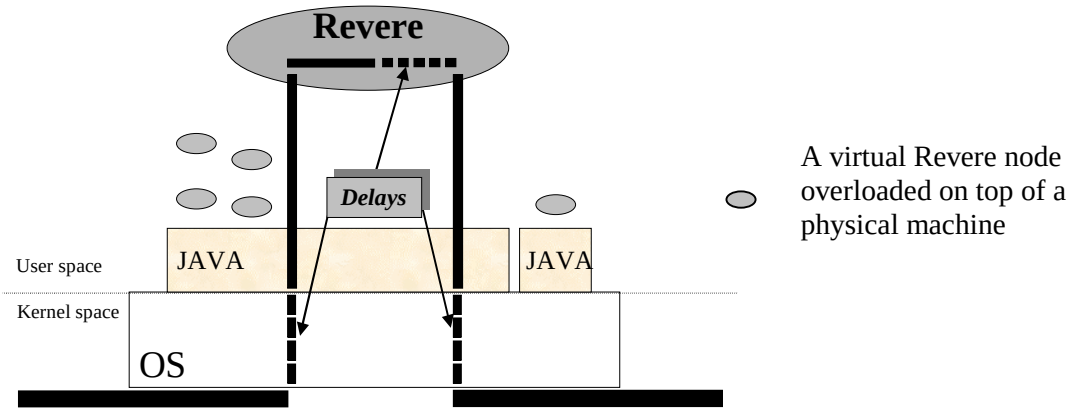


Fig 6.3 Prolonged dissemination latency in an overloaded environment

In the full system, there are two values to measure in order to determine the dissemination latency of every security update: the hop count that the update incurs when reaching a Revere node, and the forwarding order of this update at each hop (i.e. each intermediate Revere node). The hop count is used to determine the total kernel-space-crossing latency, which is the kernel-space-crossing latency per hop (to be discussed in Section 6.4.2.3) multiplied by the hop count. The forwarding order is used to determine the processing latency at each hop (to be discussed in Section 6.4.2.2). The total communication latency is calculated by summing up the communication latency in every hop.

6.3.2.3 Resiliency test phase

During the resiliency test phase, each node on the overlay network is assigned a uniform probability of failure to test how many nodes are still reachable during the dissemination procedure. The divide-and-conquer method is again used to evaluate the latency of disseminating security updates toward the remaining nodes. The dissemination latency is divided, as before, into three parts, and measurement is performed as in the dissemination phase.

RBone maintenance is turned off on purpose during this phase so that we can analyze the resiliency of a static RBone. If maintenance were turned on, every Revere node would adjust its parents if one or more parents were as detected broken; assuming the maintenance mechanism works well, almost every Revere node would then still be able to receive security updates, and even those that were not received due to a maintenance lag can be retrieved using a pulling procedure.

6.4 Results and Analysis

6.4.1 Join Latency and Bandwidth

Figure 6.4 shows the outbound bandwidth that each node incurs during the join phase, for various sizes of Revere networks. This bandwidth cost includes the messages that a node sends when joining the overlay network and the messages sent in response to the join requests of other nodes.

In this measurement, each node has to find two satisfactory parents, and must go through certain rounds of a three-way-handshake with other existing Revere nodes. The number of rounds increases logarithmically as the number of Revere nodes increases, due to the top-down recursive search of those potential parents that may still have space to accommodate a new child, as described in Section 6.3.2.1. When a new node cannot find satisfactory (efficient and resilient) parents locally, it will run the three-way-handshake with the dissemination center; if the center is full, it will then begin running the three-way-handshake with one of the children of the center; if that child is also full, it then goes to one of the children of that child; this repeats recursively. As a result, the dissemination center of an RBone incurs the largest amount of bandwidth cost, while a leaf node on the RBone probably incurs the minimal amount of bandwidth. Because of this, Figure 6.4 shows a wide variation in bandwidth cost.

Figure 6.5 shows the latency experienced by a node joining RBones of various sizes. In this experiment, each node completes the join procedure after successfully attaching itself to two existing Revere nodes that it is satisfied with. For reasons similar to the logarithmic increase of outbound join bandwidth, the latency to locate two satisfactory Revere nodes also increases logarithmically as the number of Revere nodes in an RBone

increases. The variation in join latency is at the level of dozens of milliseconds, indicating that most nodes in an RBone will incur similar join latencies.

The costs of both outbound join bandwidth and join latency are acceptable. As the number of Revere nodes varies from 250 to 3000, the outbound bandwidth per node during join phase varies from approximately 6 kilobytes to 14 kilobyte, and a new node's join latency varies from around 0.7 seconds to 1.5 seconds, both basically following logarithmic trends as the number of nodes grows.

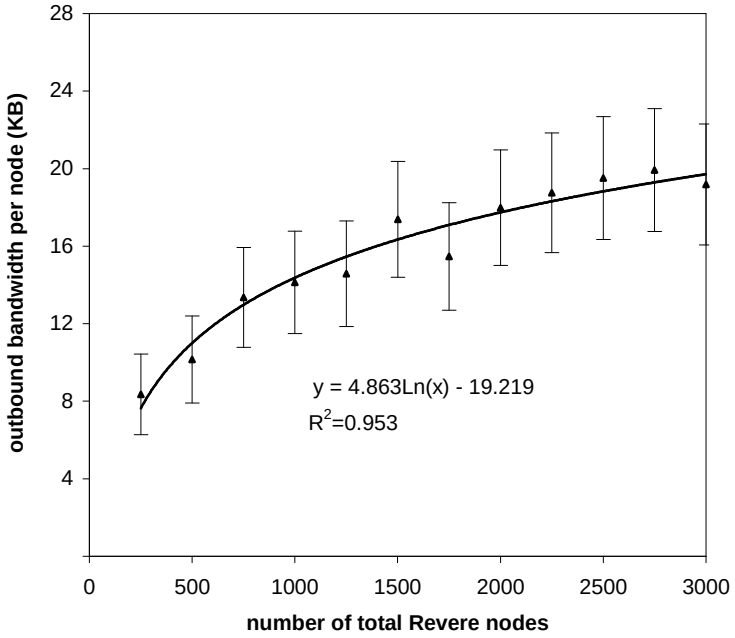


Fig 6.4 Outbound bandwidth per node during joining phase
(confidence level: 95%)

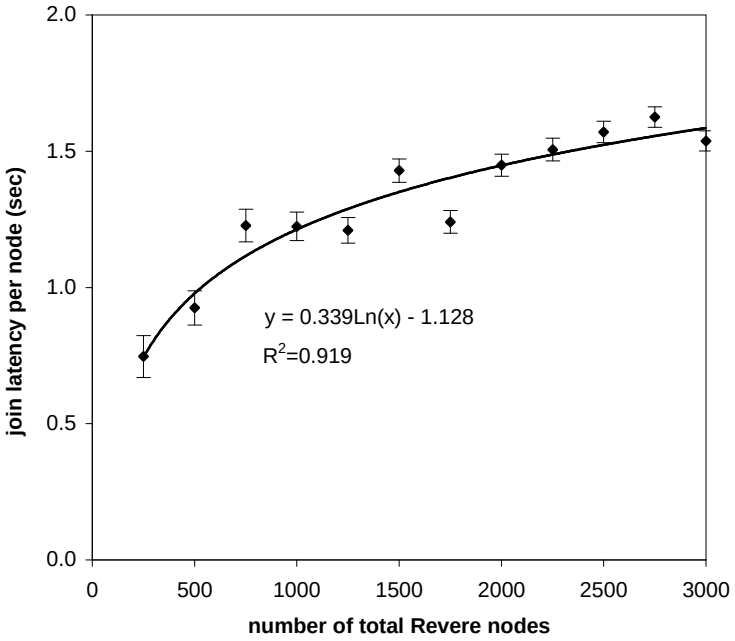


Fig 6.5 Join latency per node
(confidence level: 95%)

6.4.2 Dissemination Speed

Each security update is disseminated in a store-and-forward manner. As we described in Section 6.3.2.2, we divided the dissemination latency into three parts (step 1), evaluated each part separately (step 2), and finally combined them together in the full system to gain the overall dissemination latency (step 3).

Step 1 does not involve any measurement work. Corresponding to step 2, the evaluation of each of the three parts is described in Sections 6.4.2.1, 6.4.2.2, and 6.4.2.3, respectively. Corresponding to step 3, we describe the evaluation of the hop count in Section 6.4.2.4 and report the summed results in Sections 6.4.2.5 and 6.4.2.6.

6.4.2.1 Communication latency

As we discussed in Section 6.3.2.2, the communication latency can be calculated using the Dijkstra algorithm over the virtual topology graph underneath. With the virtual topology used for measurement, to transmit a 1-kilobyte security update, the router-to-router latency is 23.9 ms on average, and ranges from 1 ms to 70 ms.

During the measurement process, when a Revere node receives a security update from a previous hop, it will locate the router associated with itself and the router associated with the previous hop. Having knowledge of the whole virtual topology, the measurement code of this Revere node can then invoke the Dijkstra algorithm to calculate the shortest-path distance between the two routers, which we assume is the routing path taken to forward the security update from the previous hop.

6.4.2.2 Security update processing latency

In a separate experiment, we also measured the latency in processing a security update. The processing of a security update at a Revere node includes duplicate check, security check, buffering into the security update window, and forwarding to the children

of the node. The buffering operation here also includes a possible queuing delay. The processing duration begins upon the receipt of a security update. However, the processing ends at a different time for every child, depending on when a security update toward a specific child departs from the node.

During the experiment of the processing delay at a Revere node, no other Revere nodes are collocated on the same testbed machine with the tested node. This guarantees that the measured processing delay is not affected by resource contention from other Revere nodes. Meanwhile, only normal system processes are running on the testbed machine. Since in the real world every machine will run just one instance of Revere, this will give us a realistic value of processing latency.

The experiment shows that the processing delay of a security update does not correlate with the total number of children at a Revere node. Instead, it varies with the forwarding order of the update. Figure 6.6 shows that the processing delay versus the forwarding order is linear. For the first child that receives a security update from a parent, Figure 6.6 shows that the processing delay is about 1 ms, whereas it becomes nearly 3 ms in the eyes of the tenth child that receives the same update.

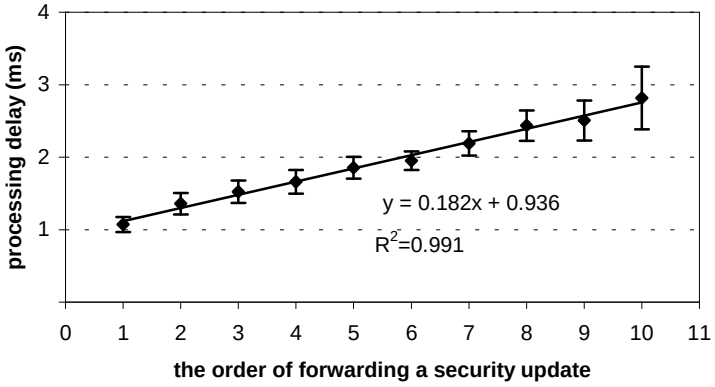


Fig 6.6 Security update processing delay at a Revere node
(confidence level for latency: 95%)

6.4.2.3 Kernel-space-crossing latency

The kernel-space-crossing delay is the time between sending a message from Revere at application level and the departure of the message from the node, plus the time between the receipt of the same message at a recipient node and the delivery of the message to Revere at application level. In another words, it is the latency from the time that a sender sends a message to the time that a recipient receives the message, but excluding the time spent on the wire, where both the sending and receiving operations happen inside Revere at the application level.

To measure the kernel-space-crossing latency, we collocated two Revere nodes on the same physical machine, and measured the interval from the time of sending a security update by one Revere node to the time of receiving a security update by another. Since no wire latency is incurred for message transmission between the two collocated nodes, this interval is used as the estimated value of kernel-space-crossing latency. Figure 6.7 shows the measured results (the spikes are caused by Java's garbage collection). With a 95% confidence interval, kernel-space-crossing latency in our measurement configuration is 674±73 microseconds.

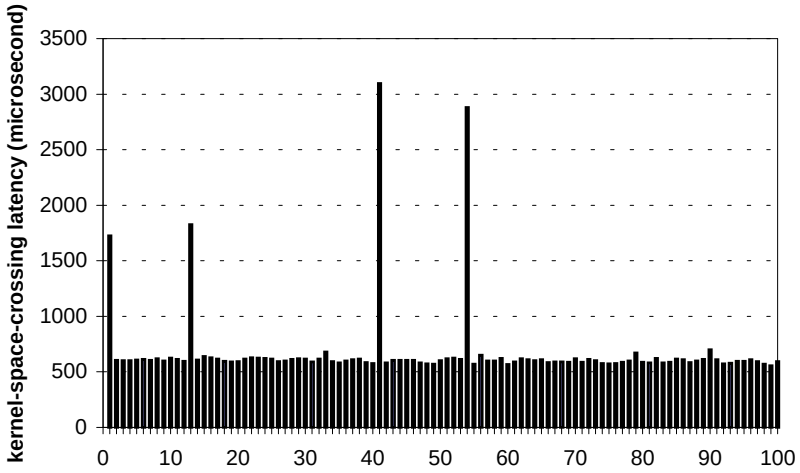


Fig 6.7 Kernel-space-crossing latency

6.4.2.4 Hop count

We measured both the average and maximum hop count for disseminating security updates (Figure 6.8). As the number of total Revere nodes varies from 250 to 3,000, the average hop count varies from 2 to 4, and the maximum hop count varies from 6 to 11.

Is such an average hop count value a reasonably good result? In those RBones measured, every Revere node had two parents. If instead, every node had just one parent, every RBone would become a tree structure, rooted at a dissemination center. For a fully saturated balanced tree, we can easily see that if the depth of the tree is 3 hops, where the root has 30 children and every non-leaf node has 10 children (the same configuration we used for measurements), such a tree can accommodate 3330 (30+30*10+30*10*10) nodes at maximum. Here, requiring every Revere node to have two parents will double the space to accommodate child nodes. Recall that if every Revere node also has

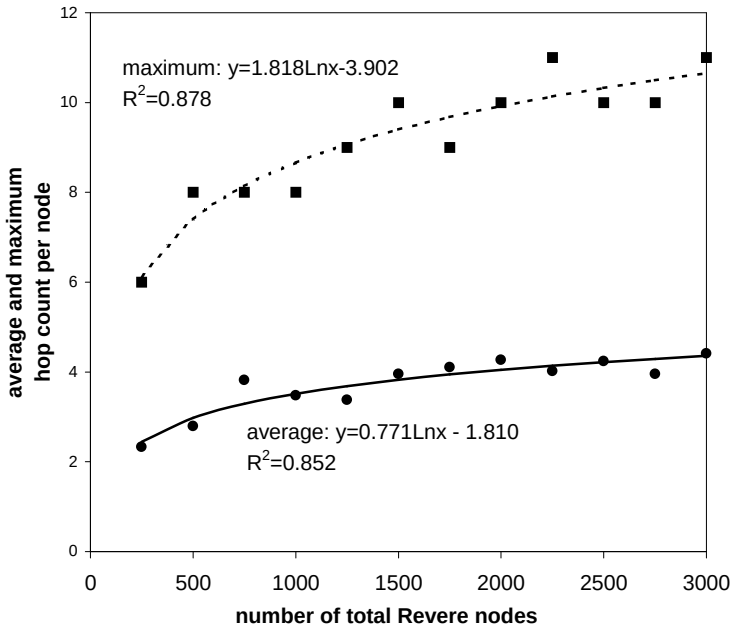


Fig 6.8 Average and maximum hop count of security update dissemination

constraints or preferences for having other Revere nodes as its parents, a 3000-node RBone with an average hop count of 4 is reasonable. This demonstrates that a Revere node in a 3000-node RBone can stay about just 4 hops away on average from the dissemination center, while still meeting those RBone formation requirements regarding efficiency and resiliency. A similar analysis can be applied to other RBones that have different numbers of Revere nodes.

Although ideally an RBone should have every non-leaf Revere node fully saturated with child nodes, and be kept balanced (just like a fully saturated, balanced tree), the maximum hop counts depicted in Figure 6.8 show that some Revere nodes could be as far as 6 to 11 hops away from a dissemination center—still considered a reasonable distance.

The best-fit trendline for a maximum hop count can easily be seen to be a logarithmic trend as the total number of nodes varies. The best-fit trendline for an average hop count is also basically logarithmic. Here, the R^2 values for both trendlines may not appear as good as one would expect; this is not a problem because a hop count can only be an integer, and this prevents the real value from being very close to the trendline.

An analysis also shows that the trendline should be logarithmic. Every node counts the latency for receiving the first authentic copy of a security update as the dissemination latency for itself. Since measurement in the dissemination phase does not assume any failure or security attack, every node must have used its fastest path to receive the first authentic copy of every security update. Therefore, it is the latency of the fastest path of every node that is measured. All the fastest paths in an RBone are rooted at the RBone's dissemination center—together they form a tree structure. The average hop count we measured should thus be the same as the average hop count of the tree. Given that the

latter follows a logarithmic trend, we can believe that the average hop count of Revere nodes in an RBone also follows the same logarithmic trend.

6.4.2.5 Dissemination latency

Based on the results in Sections 6.4.2.1 through 6.4.2.4, we then can derive the dissemination latency for every security update. Based on the measured result of kernel-space-crossing latency per hop (from Section 6.4.2.3), multiplied by the hop count measured for each security update, we obtained the total kernel-space-crossing latency for the update. Also, based on the measured result of security update processing latency from Section 6.4.2.2, combined with the forwarding order of a security update recorded at every hop, we obtained the total processing latency for the update. Recall that the communication latency of every security update was obtained using the Dijkstra algorithm during the measurement. Summing all three latencies together, we then obtained the dissemination latency for every security update.

Figure 6.9 shows both the average and maximum dissemination latency, which demonstrates a very quick response. It shows that it takes 85 ms to 300 ms on average to reach a Revere node in an RBone that has 250 to 3,000 Revere nodes. Note that the maximum dissemination latency is also the time used to reach *all* the Revere nodes in an RBone. Several outliers for maximum dissemination latency can be explained as follows: if there is one node located remotely from the dissemination center of an RBone and this node incurs the maximum dissemination latency, no matter how fast the rest of the Revere nodes receive security updates, the maximum latency will be based solely on the dissemination latency of this single node.

More interestingly, Figure 6.9 shows that both the average and the maximum dissemination latency closely follow logarithmic trends. If we use these trends to predict

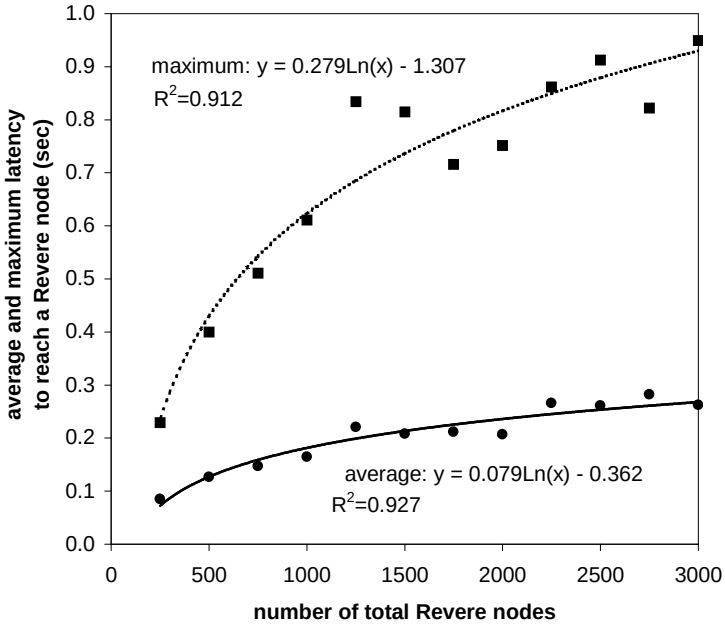


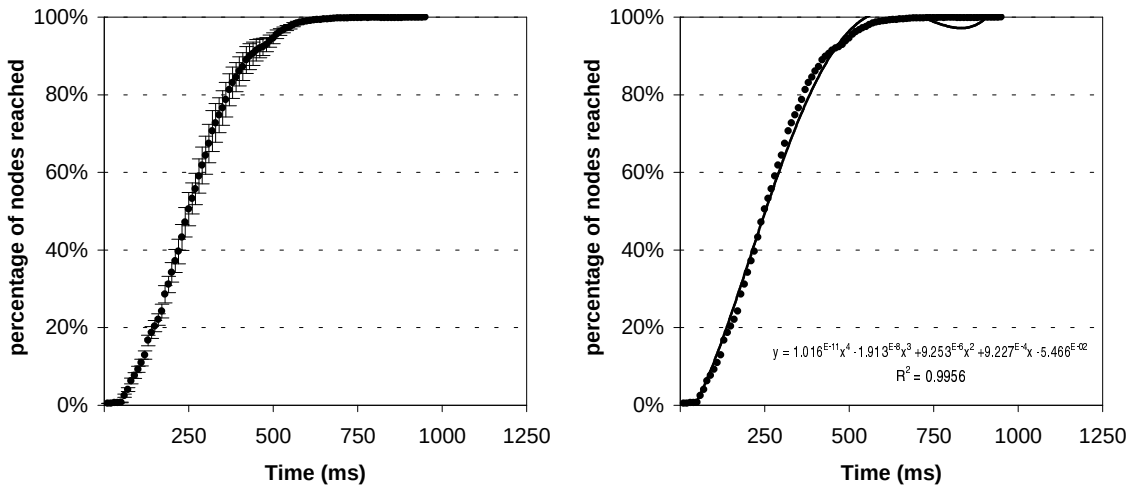
Fig 6.9 Average and maximum security update dissemination latency
(confidence level for average latency: 99.9%)

the dissemination latency in an even larger scale, encouraging results can be gained. For example, reaching a Revere node in a 100-million-node RBone will only take approximately 1.10 seconds on average, and the maximum dissemination latency is 3.83 seconds. Certainly, there are many issues to be addressed when extrapolating both trendlines for a larger scale. We will discuss this in Section 6.5, Open Issues.

6.4.2.6 Dissemination coverage

It is also worthwhile to ask what latency is needed to reach a certain percentage of nodes in an RBone. Or, conversely, to ask what percentage of Revere nodes are reached at a given time?

We obtained the dissemination latency of each individual node and then derived the percentage of nodes covered as the dissemination proceeds. Figure 6.10 shows the dissemination coverage over time for a 3000-node dissemination. In this case, 100% of



(a) Coverage with confidence level of coverage: 99%

(b) The same coverage graph as that in (a), but with a polynomial trend

Fig 6.10 Security update dissemination coverage for a 3000-node dissemination

the nodes are reached in a short time (less than 1 second). Figure 6.10 (a) shows the coverage with a 99% confidence interval, and Figure 6.10 (b) shows the same coverage graph but with a polynomial trendline. The trendline suggests that as time goes by, more Revere nodes are reached at a polynomial speed (except that the tail part does not follow closely, as shown in Figure 6.10 (b)).

Clearly, the speed of reaching 100% coverage from 0% coverage is not constant. In particular, the dissemination coverage has a long "tail" as shown in Figure 6.10. Although at time 610 ms, 99% of nodes have already been reached, it is not until time 950 ms that 100% of nodes are reached. If what one really cares about is not necessarily the full-coverage latency, even lower latency can be derived. We look further into this issue as follows.

Based on the dissemination latency to reach every Revere node in an RBone, we obtained the latency for reaching a certain percentage of nodes in the RBone. Figure 6.11

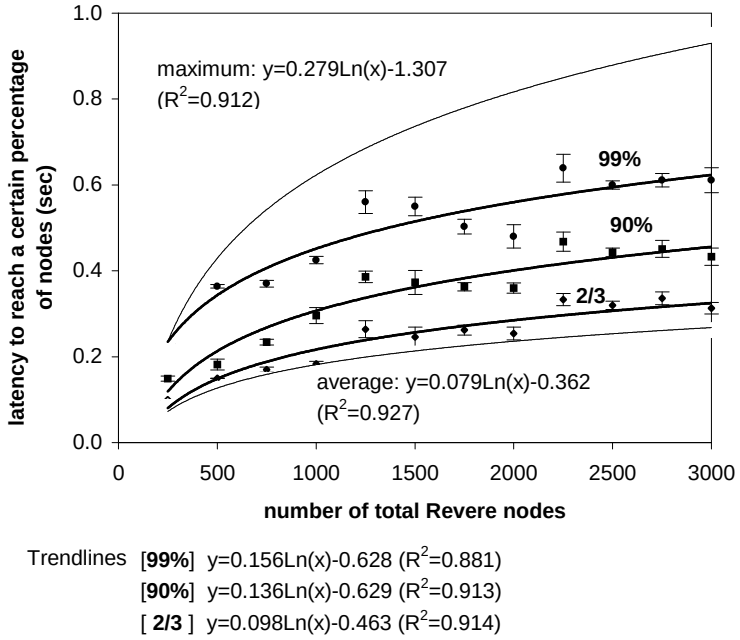


Fig 6.11 The latency to reach 99%, 90%, and 2/3 of Revere nodes in an RBone
This is also compared with Figure 6.9's maximum and average latency to reach Revere nodes, whose trendlines are also shown in this figure.

shows the latency for reaching 99%, 90%, and 2/3 of Revere nodes in an RBone, where the total number of RBone nodes varies from 250 to 3,000. Not surprisingly, all three trendlines for the 99%, 90% and 2/3 coverage closely follow the logarithmic trend. If we use the trendlines in Figure 6.11 to predict the latencies in reaching 99%, 90% or 2/3 of the nodes in a 100-million-node RBone, it will take about 2.246, 1.876 and 1.342 seconds, respectively.

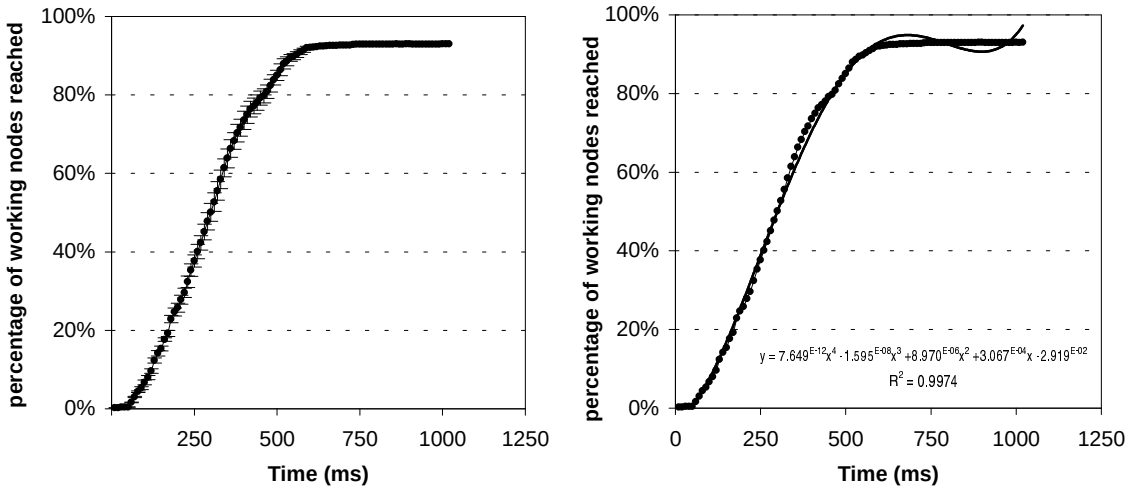
6.4.3 Dissemination Resiliency

During the resiliency test phase, an RBone's resiliency was tested using different probabilities that nodes would be broken. After a dissemination center broadcasts a security update, every Revere node, once visited by the security update, will determine whether it should emulate a broken node or a working node by asking a common random

boolean server. The random boolean server answers by comparing a newly generated random number r ($0<r<1$) with the given broken probability. Clearly, a visited working node is also a reached working node.

We regard the rest of the nodes that are not visited as *unreached working nodes*. There are, then, totally three types of nodes: broken nodes, reached working nodes, and unreached working nodes. The former two are visited nodes during measurements, and the ratio of broken nodes over the total number of visited nodes should be approximately the same as the assigned broken probability.

We use the ratio of reached working nodes over the total number of working nodes to measure the resiliency of an RBone. In this section, we focus on 3000-node RBones. Figure 6.12 depicts the resiliency characteristics of the same RBone reported in Figure 6.10; the difference is that now every (visited) Revere node has a 16% probability of being broken.



(a) Coverage with confidence level of coverage: 99%

(b) The same coverage graph as in (a), but with a polynomial trendline

Fig 6.12 Resiliency test for a 3000-node dissemination with a 15% node failure

With the failure of as many as 15% of the total nodes (resulted from the 16% broken probability of visited nodes), a high percentage (93%) of the working nodes are still able to receive security updates without readjusting the structure of the dissemination overlay network. Figure 6.12 (a) shows the coverage over time with 99% confidence. Compared with Figure 6.10 on the same RBone but without node failure, now the dissemination latency is longer. However, this is still a very fast response, particularly with a fairly high probability of node failure. Figure 6.12 (b) shows that the dissemination latency matches smoothly with a polynomial trend.

Measurement shows that the 3000-node RBone is resilient to small node-broken probabilities (with node-broken probability lower than 2%, 100% working nodes can still be reached). Corresponding to four different higher failure probabilities, Figure 6.13

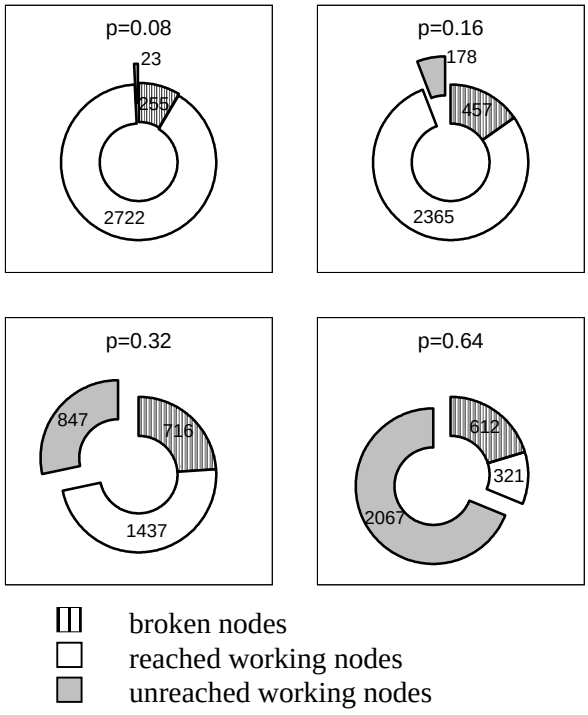


Fig 6.13 Resiliency test with different node broken probability on a 3000-node RBone

shows the resiliency test results in four doughnut charts (the pulled-out slice represents the unreached working nodes). This demonstrates a very resilient RBone. Recall that in measurement, every node has two parents; we believe even better resiliency will result if additional parents are allowed.

6.5 Open Issues

6.5.1 More Thoughts on the Overloading Approach

The "overloading" approach to measuring large-scale distributed systems requires that multiple nodes of a distributed system, if collocated on a physical machine, can still perform correctly. In practice, however, distributed systems are typically designed with the presumption that a single instance of the software executes on each physical node. Slight modifications to a distributed system may be necessary in order to measure it using this approach. As we pointed out earlier, systems that use an IP address as a node name will require modification.

Theoretically, it might also be possible to build a common framework based on this approach to support measurement of differing distributed applications, and a specific distributed system can be measured by simply plugging it into such a framework. Designing an interface between the framework and the application being measured then must be carefully considered.

Another issue is the scalability of this approach itself. Given that multiple nodes under this approach can contend for resources of the same physical machine, some resource locking mechanism is needed to obtain accurate results. An example of this approach is the token mechanism used in measuring the join performance of Revere. However, this technique slows down the measurement process. The token-controlled

mechanism used to measure joins in Revere, for example, required about 100 minutes of measurement for 3000 nodes (with the configurations described in Section 6.3.1).

6.5.2 Performance Understanding at Larger Scale

In previous discussions (particularly Sections 6.4.2.5 and 6.4.2.6), performance at larger scale is predicted by extrapolating a trendline. The question is, will a trend that is derived from results at smaller scale still apply to a larger scale? Is there a point that the curve could suddenly change its inclination?

More measurements at larger data points can help promote more confidence regarding a trend, but since there are almost always values at even larger points that one cannot measure (in Internet scale), one cannot measure infinitely. One also has to rely on the analysis of the system behavior to understand its characteristics at larger scale.

6.5.3 Challenges from the Real World

A real-world environment is always more complicated than a measurement environment, and thus harder to measure and understand. Even though the measurement of Revere used the real Revere code, there could still be factors that cannot be captured in the measurement. Some of those factors are hidden and hard to discover and determine, and some of them are very difficult to manipulate due to the tremendous complexity. The following is a partial list of those challenges:

First, the real world is heterogeneous. We assumed a homogeneous setup in measuring Revere. However, taking a snapshot of the Internet at any moment, one would see a highly heterogeneous composition in almost every aspect (even the Internet Protocol, regarded as the only invariant by many, is represented by both IPv4 and IPv6) [Floyd *et al.* 2001]. From the viewpoint of Internet elements, autonomous domains are different in that they enforce different routing and security policies; routers are

heterogeneous in that they run different routing protocols with different capabilities; links are heterogeneous in that they have different capacity and quality; hosts are different in that they have different performance on top of different platforms with different security and mobility constraints.

The heterogeneity can also be exemplified by two common practices over the Internet: routing and TCP. Routing is often asymmetric. According to [Paxson 1996], a path through the Internet in 1995 visited different cities in each direction 50% of the time and different autonomous systems 30% of the time. TCP, while being widely used, has more than 400 different implementations and versions as identified using the "fingerprinting" technique (a technique that compares protocol behavior in response to different input) [Fyodor 2001].

Second, the real world is always moving. The Internet has been drastically changing since its inception, including changes to its various elements, protocols, and traffic. We plotted Figure 6.14 according to data reported by netsizer.com as of May 13, 2002. Interestingly, we can observe that, beginning in January 2002, the increase to

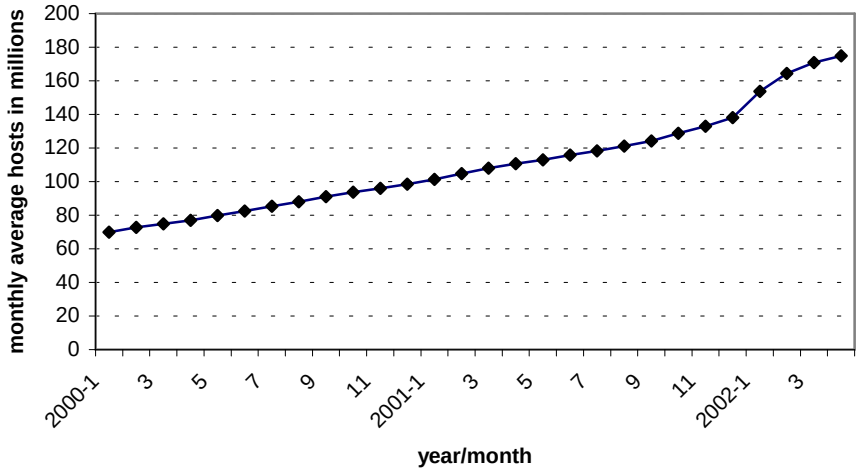


Fig 6.14 Monthly average Internet hosts in millions

speed switches from linear to logarithmic. While it is interesting to watch this growth trend, it indicates that predicting the growth of the Internet is clearly not an easy task. For example, anyone trying to predict Internet growth in the year 2001 could have easily made a big mistake in predicting growth for the year 2002, if he had based his prediction on a linear growth rate.

Traffic is also non-deterministic and highly dynamic. For instance, [Floyd *et al.* 2001] reports that USENET traffic has been exponentially increasing since 1984. Experiments with FTP traffic, web traffic, etc., also show this kind of dynamics. Congestion control, traffic control, etc., on the other hand, try to smooth or regulate the traffic to better utilize Internet resources, further making it difficult to capture traffic charactristics.

Third, the Internet is a large-scale target to understand. Due to its great success, the Internet has grown into an immensely large entity. While it is already difficult to predict Revere performance at larger scale (as discussed in Section 6.5.2), the unpredictability caused by heterogeneity and dynamics at larger scale makes this even harder.

6.5.4 Measurement of an RBone's Physical Layer

So far Revere is only measured in the application layer. While an RBone is an application-level overlay network that connects Internet hosts, its physical-level properties are still much less understood.

One important question is: how disjoint are a Revere node's multiple delivery paths? Although Revere is designed to have them as disjoint as possible, this is mainly done at application (logical) level. If there were a router that was located on all delivery paths (which are seemingly disjoint at the logical level) and an attacker could locate this router,

the attacker could still try to launch interruption threats or other attacks by breaking into this router. This is not a problem when the attack model only deals with Revere nodes; however, this requires a strong trust model of the underlying routing infrastructure. Ideally, one would hope that if delivery paths are disjoint at application level, their overlapping degree at the physical (router) level is also low. Clearly, more measurements are needed to answer the question.

Another important question is: would a particular physical link on the Internet be overloaded by Revere control messages or security updates? At application layer, every Revere node only has a small number of parents and children, and only communicates with those nodes most of the time; as a result, no Revere (logical) link will be overloaded. However, it is unknown whether those Revere links might actually share a common physical link. If a physical link is overwhelmed by a huge amount of Revere messages, the performance of the system may degrade rapidly.

6.6 Conclusions

6.6.1 The Overloading Approach

As more distributed systems run at Internet scale, understanding the performance of a system at large scale is important. Unfortunately, it can be difficult to measure a system that consists of very large numbers of nodes that are part of a large-scale network.

Without actual deployment, measurement of a large-scale system can be performed in two ways: the first is simulation and the second is the overloading approach. Simulation is a popular approach for large-scale systems. However, since a simulation does not typically use the actual software and cannot accurately emulate all environmental factors, it is very difficult for simulation tools to capture all the real effects of the system.

Our overloading methodology collocates a large number of nodes of a distributed system on a machine, while still allowing each node to run the real software. This methodology can accurately report those metrics that are invariant with respect to overloading, and can minimize those inaccuracies introduced due to overloading and resource contention.

Meaningful results can be obtained. We demonstrated this using the overloading approach for a security update dissemination system. While the measurements reported in this chapter correspond only to up-to-3000-node networks, the results were obtained using only 10 nodes. We believe that Internet-scale results can be obtained using only a few hundred or a few thousand nodes. In addition, we believe that this approach can be further generalized into a common framework to support measurement of different distributed systems.

As discussed earlier, the overloading approach shares many of the validation problems that simulation also experiences, whereas it has the singular advantage of running the real software.

6.6.2 The Performance of Revere

Encouraging results were obtained for all six metrics—dissemination bandwidth, RBone maintenance bandwidth, join latency, join bandwidth, dissemination latency, and dissemination resiliency. A new Revere node can quickly join an RBone after spending a small amount of bandwidth. An RBone can be maintained using a reasonable amount of bandwidth, and it can also be made resilient to node failure or subversion. Moreover, with a reasonable amount of bandwidth cost, security updates can be delivered to the whole RBone quickly.

While Revere provides a dual mechanism for delivering security updates, it is mainly the pushing mechanism that was measured and discussed in this chapter. If every Revere node has the same configuration as the testbed machine used (whose platform and performance is that of a common PC in today's network), dissemination can be achieved in seconds. It also shows that a high percentage of Revere nodes can be reached much earlier than before *all* nodes are reached. This push model is superior to the pull model that is in current use, which has to rely on high-frequency pulling to keep a machine updated (at the cost of bandwidth).

The measurement further shows that an RBone can be made resilient to failures and attacks. It proves that a self-organized resilient RBone can be made robust without employing powerful servers over the Internet. In the 3000-node RBone tested, for instance, even with a 15% node failure, 93% of remaining nodes can still receive security updates. Note that in the measurement, every node is configured to have only two parents. We believe that if Revere nodes choose to have more parents, better resiliency results will be realized.

The measurement also demonstrates that Revere scales well. The join bandwidth cost, the join latency, the (average and maximum) hop count, the (average and maximum) dissemination latency, and the time to reach a certain percentage of Revere nodes, all appear to closely follow logarithmic trendlines as the total number of Revere nodes increases.

In conclusion, the Revere overlay network is fast, resilient, lightweight, and affordable.

CHAPTER 7

Related Work

7.1 General-Purpose Distribution Services

Viewed in the most general context, Revere fits within the broad scope of information distribution over the Internet. In this section we look into those general-purpose distribution services, including preliminary techniques (unicasting, broadcasting, flooding, etc.), IP multicasting, application-layer protocols, email, replicated data management, content-delivery networks, and some commercial products.

Many of the low-level needs of the Revere system can be met by use of these and other existing message delivery and network security services. Recall that Revere allows two Revere nodes (parent and child) to negotiate the delivery mechanism for forwarding security updates. On the other hand, these services do not solve the entire problem that Revere addresses.

7.1.1 Preliminary Techniques: Unicasting, Broadcasting, and Flooding

The simplest approach to information distribution is to unicast, but it is not scalable to unicast security updates to millions of nodes from a dissemination center, one by one. There are tens of millions of machines connected to the Internet, and each machine is a potential participant. Because the Internet is ever growing, it is impossible for a single machine, or even dozens of machines, to store global knowledge concerning all potential participants. Even were this feasible using powerful machines, the task of keeping information up to date is daunting. The unicast approach, which is based on centralized management, is thus difficult, if not impossible. Further, high scale ensures that

significant numbers of nodes will be disconnected at the moment a security update is being disseminated. A unicast-based approach must also include features to make updates available to those nodes that missed them during dissemination.

Existing Internet broadcast mechanisms that work at the IP level use a best-effort delivery [RFC919], [RFC922]. They do not guarantee delivery, and they have no concern for security. Further, they are not designed to deal with the kind of scale that Revere will handle. They are meant for broadcasting to subnets or small collections of subnets. However, the number of subnets over the Internet is still a large number, and the broadcasting mechanisms must reach all those subnets that have recipients separately, still one by one.

Multi-network broadcasting uses a "broadcast repeater" to forward broadcast messages over IP networks [RFC947]. The forwarding address is read from a configuration file, which is not adaptive. The addresses of each repeater's downstream nodes are fixed. Revere needs to deal with more dynamic networking. Again, this solution assumes the correct participation of all nodes, particularly of all broadcast repeaters, and does not include provisions for security.

A simple flooding algorithm might be used that does not require maintaining any state or topological information [Goldreich *et al.* 1992], [Bertsekas *et al.* 1992], [Bauer *et al.* 1992]. Each node that receives the broadcast message transmits on all interfaces except the incoming one. After sufficient steps, the message will reach all nodes. Flooding methods tend to be inefficient in their use of bandwidth, because they usually send a copy of the message over every possible link. Also, they usually provide no support for disconnected nodes, and assume the correct participation of all nodes. Clearly, using a flooding algorithm at Internet level is not feasible at all.

7.1.2 IP Multicasting

Multicast services, such as MBONE [Deering 1989], [Macedonia *et al.* 1997], [Venkateswaran *et al.* 1997], allow dissemination of information to a large number of users. Like Revere, multicast is receiver-based, but typical multicast services offer no guarantee of delivery [Floyd *et al.* 1995] and have no redundancy. A great deal of research has been performed on multicast and many different multicast systems handle some Revere-like issues. However, no existing multicast service currently handles all of those problems, nor can one be made extensible in a relatively simple manner in order to handle them.

Multicast services typically use tree-structured routing, implying a single path from a disseminating machine to any recipient. These services are thus not resilient to attacks on individual links and nodes. If a node on the tree is broken, all the descendents of this node will be disabled from receiving the security updates.

Furthermore, most existing routers do not provide multicast routing capabilities, limiting the use of multicast protocols. If Revere is to be deployed into the existing Internet and supply services even to legacy systems that do not provide multicast, it cannot rely on the presence of such protocols.

Reliable multicast protocols [Chang *et al.* 1984], [Kaashoek *et al.* 1989], [Moses *et al.* 1989], [Yavatkar *et al.* 1995], [Levine *et al.* 1996], [Lin *et al.* 1996] seek to ensure that all receivers get complete and correct information, typically by acknowledgments for all packets, which leads to an acknowledgment implosion problem. The use of negative acknowledgments allows many multicasting systems to avoid acknowledgment implosion, but this technique is not suitable for Revere's needs, because a Revere node cannot assert that it missed a disseminated security update or did not. Security update dissemination is not a periodic behavior, and a dissemination center sends out updates

whenever new updates are available. Repair-request methods are also used by some multicasting systems to solve this problem, but, like negative acknowledgments, they require the receiving nodes to be aware that they missed something. Last, reliable multicast mechanisms are designed for packet loss or damage due to transmission errors, not for loss or damage due to interruption threats.

Multicast systems vary tremendously in their details. In a typical multicast system, a multicast tree (shortest path tree or Steiner tree) is built for efficiently sending a message to a given set of destinations [Garcia-Lunes-Aceves 1993]. The tree can be quite unbalanced in terms of performance for different senders. An overlay hypercube topology was also used for multicasting, significantly improving the performance for any sender. [Moser *et al.* 1997] uses a SecureRing protocol to protect against Byzantine failures, an unsuitable solution (and overprotection) for the scale and requirements of Revere. Other reliable multicast research includes consistent totally ordered delivery of data to all receivers [Birman *et al.* 1991] [Whetten *et al.* 1995], handling large numbers of acks [Pingali *et al.* 1994], using minimal network resources [Yavatkar *et al.* 1993], and managing the multicast groups [Yavatkar *et al.* 1995], [Liebeherr *et al.* 1997]. These objectives are different from rapid dissemination of a small amount of data sent to all nodes.

In a word, IP multicast still faces many problems for deployment at a large scale, and cannot distribute to all recipients unless they are all connected simultaneously. Reliable multicast is better, but it mainly handles packet loss caused by transmission errors, not loss caused by attacks such as interruption threats.

7.1.3 SMTP, NNTP, FTP, HTTP

At a higher layer, smtp, nntp, ftp, http, etc., all provide certain distribution capabilities, but it is difficult to tailor these capabilities to meet the challenges of providing a successful security update dissemination service. None of these provide a resilient network to address man-in-the-middle delivery threats, none provide both pushing and pulling mechanisms for best large-scale delivery coverage, and none fully address security.

7.1.4 Email

Email has been used as a carrier for alerts [Frank 2000]. Unfortunately, email could also be a carrier for malicious functions, and an email that carries alerts of a malicious function often arrives later than an email that carries the malicious function. This makes email an unrealistic solution for delivering important security information. For example, when facing the love bug in year 2000, panicked federal agencies completely shut down their mail servers, making them unable to receive emailed alerts from the federal computer incident response capability team. The security team ended up using phones and faxes to send alerts, clearly a scheme not suitable to Internet-scale information dissemination.

7.1.5 Replicated Data Management

Research in replicated data management, particularly optimistic peer replication, provides insight into methods for ensuring that different sites share the same view of the data. Client/server solutions [Satyanarayanan *et al.* 1990], [Kistler *et al.* 1991] and primary copy solutions [Liskov *et al.* 1991] are less relevant, since they cannot allow update dissemination between arbitrary participants (a requirement of the Revere system).

Ficus allowed multiple replicas of files, any one of which could initiate an update and any pair of which could exchange updates [Guy *et al.* 1990], [Page *et al.* 1998]. This functionality is close to what would be required for Revere. Other work done in the Ficus project addressed issues of ensuring consistent views of changing replicated data [Goel 1996]. Truffles, a related project, provided some forms of security for optimistic replicated file systems [Reiher *et al.* 1993]. Rumor provided a more portable version of the functionality of Ficus [Reiher *et al.* 1996], and also more directly addressed issues of mobility and replication. Recent research on the Roam replication system has dealt with replication and mobility at higher scales, with hundreds or thousands of replicas [Ratner 1998], an issue of great relevance to Revere. Still, no replicated file system research has claimed to support scaling to millions of replicas. Simulations of the behavior of replicated file systems have offered important insights into the proper way to disseminate shared data among large numbers of participants [Wang *et al.* 1997]. This and other relevant replication research [Birman 1985], [Alonso *et al.* 1989], [Hisgen *et al.* 1990], [Badrinath *et al.* 1992], [Golding *et al.* 1993], [Danzig *et al.* 1994], [Demers *et al.* 1994], [Gray *et al.* 1996] provide insights on data consistency and dissemination issues in Revere, but these works are directed at solving much more general problems than is Revere. Revere uses simpler, lighter-weight solutions than the file replication systems mentioned above.

7.1.6 Content-Delivery Networks

Much research has also been done on content delivery networks (CDN), using distributed caching or overlay techniques. InformationWeek compared seven networks: Adero, CacheWare, Cidera, Digital Island, epicRealm, iBeam, and Mirror Image Internet [Patrizio 2000].

- Adero focuses on E-commerce application services, with servers in more than thirty countries. Its GlobalWise Applications and GlobalWise Commerce Business are designed to work with a company's existing network to interact with worldwide customers.
- CacheWare specializes in content distribution and caching from an origin server to edge servers. Its CacheWare Content Manager takes the load off an origin server by acting as the intermediary between origin and edge servers. Rather than requiring each edge server to contact the origin server, CacheWare pushes updated content to edge servers.
- Cidera's world-wide network is satellite-based and specializes in transporting data streams. In addition, it also offers static content caching and Usenet, allowing customers send huge files without choking network servers.
- Digital Island was the first company to offer content delivery in 1996. It has 160 access points in 25 countries, and provides content-delivery and application services for secure transactions and network awareness. It also uses Traceware, a product that uses IP addresses to predict client geographical locations. In this way, its customers can deliver targeted content in any region.
- EpicRealm focuses on the business-to-business market and lets customers be served by local servers, regardless of their locations. It caches static and dynamic content, database-driven content, and even encrypted content.

- iBeam specializes in streams via satellite rather than terrestrial lines, reducing the number of hops to transmit the stream and thus reducing packet loss. A terrestrial transmission can go through as many as 20 hops, while iBeam sends the stream from the source straight to the satellite, then back to edge servers at ISPs and major data centers. The data travels over only the last mile to the user on landlines.
- Mirror Image Internet specializes in caching technology and is building what it calls a "content access points" network designed for integrating into existing data centers and accelerating the mirroring, caching, and delivery of content.

Unlike Revere, where security updates are usually of small size and low frequency, CDN must handle large blocks of data. Because of this, Revere structure is fundamentally different from a CDN structure, where the latter does not support parallel redundancy of information delivery.

7.1.7 Commercial and Research Products

Many commercial products and research projects support data broadcasting [Bannister *et al.* 1997]. For example, SATX is an asynchronous communications program designed to transfer binary files over a data broadcasting network through direct broadcast satellites (DBS). More sophisticated schemes maintain topology information to minimize resource wastage and avoid duplicate messages. Broadcast mechanisms generally do not guarantee delivery to all sites, or handle disconnected and mobile nodes, or authenticate messages.

Products such as Pointcast send individually customized information to large numbers of users periodically. It sends different information to different users using standard

Internet protocols, and is essentially a centralized approach. Clients of Pointcast cannot benefit each other by forwarding information.

Salamander is a wide-area network data dissemination substrate designed to support push-based applications [Malan *et al.* 1997]. Salamander uses a dynamically constructed tree of distribution servers to push data from suppliers to clients. Salamander is not meant to provide security, nor is it meant to handle temporarily disconnected nodes.

The Clearinghouse project at Xerox PARC [Demers *et al.* 1987] addressed the problem of efficiently disseminating name-to-address mappings for a corporate electronic mail environment spanning several hundred LANs scattered around the world. A server on each LAN hosted a replica of the mapping database (the Clearinghouse), and could independently generate updates. The Clearinghouse algorithms propagated updates using several different mechanisms, the most important and effective being "rumor mongering," a constrained flooding/gossiping approach with relatively low overhead and high probabilities of effective, reasonably quick update distribution. While this work has many surface similarities to Revere, it handles a different problem and has other dissimilar characteristics. First, Revere's scale is several orders of magnitude larger, and it must handle a more rapidly changing network topology than does Clearinghouse. Frequent medium-term disconnections of nodes are the norm, and Revere must deal with node and link failures on-line and automatically. Second, unlike Clearinghouse, Revere cannot trust all its nodes completely. Third, Clearinghouse's operational constraints allow for a daily six-hour window in which to propagate updates among replicas. Revere must share all resources with normal demands, and it must propagate much more rapidly than daily in some predefined period. Last, Revere faces a much larger heterogeneity problem than Clearinghouse faces.

Other commercial products that provide information dissemination services include BackWeb, Ifusion, InCommon, Intermind, Marimba, NETdelivery, and Wayfarer. Many of these commercial systems require clients to periodically poll the servers for new data, fetching it if available. This approach has performance problems at high scale, especially if pulling is frequent. But if pulling is infrequent, data dissemination is slow.

7.2 Special-Purpose Distribution Services

Given that there are many special-purpose distribution services, one might think of tailoring or extending those services for security update dissemination. Unfortunately, from those special-purpose distribution services surveyed, this is not an easy task. In the following sections, we describe several such distribution services, including virus signature distribution, NTP (network time protocol for clock time synchronization and distribution), event notification, key distribution, and software distribution.

7.2.1 Virus Signature Distribution

When considering application purpose, virus signature distribution, as one of the earliest practical purposes for security information dissemination, is probably the most similar to Revere service. Many industrial groups devote substantial efforts to identifying and combating new viruses, including Symantec, IBM, and McAfee Associates. For example, Symantec has anti-virus tools for protection against known viruses, and takes aggressive actions to find new viruses as they occur, producing detection and repair mechanisms for them soon after they are discovered. These groups offer signature distribution services to their customers, allowing a customer to download newly discovered virus signatures and response mechanisms on demand.

This strategy for virus signature distribution does not make use of the existing network except in the most trivial way. Each participant must directly contact the virus

protection group's site to receive updates. Updating is typically done in a pull fashion, either when scheduled by the user's machine or on command. This often fails to instantaneously keep a user's machine updated, unless the user probes very frequently, which unfortunately incurs high bandwidth cost; the web site would also be overwhelmed by requests from a multitude of users.

At IBM, researchers are developing anti-virus technology based on human immune systems. In their approach, a potentially infected computer sends a suspicious file to a central site where it is analyzed for the purpose of determining a virus signature. This signature is sent back to the computer that found it, presumably including an antidote to the virus, as well. This antidote is then sent from computer to computer via a simple distributed dissemination mechanism designed for a local area network. This anti-virus system is not designed to be secure from attacks. The authors have not yet reported on extending it to handle mobile and disconnected systems [Kephart *et al.* 1997].

Some groups set up central servers to automatically broadcast new virus signatures to every individual user, but difficulty in managing user records at the central servers grew quickly as more users participated. Peer-to-peer technology has been used recently to address some of these problems, where information can be forwarded along a chain of recipients [McAfee Rumor]; however, the design technology to handle disconnected nodes, strengthen security (including combating interruption threats), and maintain the chains has not been reported. Revere, in addition to supporting the pulling mechanism, provides a non-centralized push model that better addresses efficiency, resiliency and security.

Revere is able to ensure wider, faster distribution of virus signatures and similar security information while proving strong security and resiliency.

7.2.2 NTP—Network Time Protocol

The Network Time Protocol solved a problem with some similarities to problems that Revere addresses [Mills 1991]. The Network Time Protocol (NTP) ensures that large numbers of networked sites substantially agree on the current time. The designers of the protocol foresaw possible security problems, and dealt with them in the protocol.

The NTP disseminates clock information to a large, diverse Internet system over subnetworks operating at speeds from mundane to light-wave speed. Like Revere, the messages in question tended to be small. NTP uses backup paths to achieve robustness, echoing Revere's use of redundant delivery paths. NTP supports multiple service classes, based on the needs of particular nodes. NTP addresses security issues including address filtering for access control, authentication via digital signatures, protection from untrusted time servers by data filtering and peer selection and combining algorithms.

However, the problem that NTP solves is different from Revere's in important ways. NTP requires manual configuration, while Revere must do automatic configuration. Moreover, a backup path can evolve into a primary path in NTP, a strategy that works better in NTP than it does in Revere, since an NTP node has reasonable expectations of when NTP messages should arrive, and can switch paths when expected messages do not arrive. Revere messages are unpredictable, and a Revere node cannot locally distinguish between a situation where no Revere messages are being sent and a situation where subversion prevents delivery of messages.

Also, NTP does not require retransmission of missed messages. Time updates are, by their nature, ephemeral. A missed update is of no value shortly afterwards, so disconnected nodes have no need to acquire them.

7.2.3 Event Notification

Event notification services, which usually adopt a centralized approach, focus on different issues and often view the mapping between event subscribers and event publishers as a key issue [Krishnamurthy *et al.* 1995] [Gruber *et al.* 1999].

[Cabrera *et al.* 2001] proposes another event notification service called Herald. Similar to Revere, Herald is designed to operate correctly in the presence of numerous broken and disconnected components. Also similar to Revere, Herald also views machines as a federation of nodes within cooperating but mutually suspicious domains of trust. In particular, Herald addresses scalability, an issue less addressed in previous works on event notification service. One key difference between Herald and Revere is the delivery of information. Herald does not provide redundant delivery (actually it tries to avoid it), and Revere views redundant delivery as a fundamental basis for security update dissemination.

7.2.4 Key Distribution

Key distribution mechanisms also have some relationship to Revere's design. The main goal for key distribution, however, is secrecy of the keys, while in many cases secrecy is of secondary importance for Revere. Quick dissemination and high availability will often be more important for Revere than for many key distribution facilities. Generally, key distribution systems do not operate at the scales envisioned for Revere.

7.2.5 Software Distribution

Many software vendors have adopted the Web for software distribution and update distribution, relying primarily on user pull techniques where a user downloads the

software from the Web. Users need to pull the new releases individually, which results in heavy traffic and delay when, for example, Microsoft releases upgrades to its browser.

Several commercial products allow automatic updates of popular software, such as Symantec's TuneUp [Symantec TuneUp] and McAfee's Oil Change [McAfee OilChange]. TuneUp monitors which programs a user has, and automatically downloads and installs the updates when they become available. Oil Change software makes clever use of push technology to notify users automatically when updates are available. The software can be updated with a one-button push from the user. Both products are designed to facilitate software updates for PC users.

In some overlay networks, control software can be updated automatically. For example, in Metricom Ricochet wireless networks, the gateway software is updated automatically when new versions are available. Here, the sites to be updated are controlled by Metricom, and the time delay is not critical.

All of these automatic software update schemes are designed for ease of use and cannot provide reliable transfer of data. Their concerns with security primarily relate to authentication of the server directly to the client, or vice versa. They also do not take advantage of broadcast medium in some local networks, as the downloads are accomplished using point-to-point TCP protocol. Using such automatic distribution mechanisms in Revere to distribute security information will not yield a rapid, reliable, and secure mechanism.

7.3 Information Delivery Structures

7.3.1 Overlay Networks

Revere's RBone overlay network is comparable to various self-organizing overlay networks that are also composed of Internet end hosts. *Yoid*, for example, tries to build a

general architecture for information distribution, including a tree topology for content distribution and a mesh topology for control information distribution [Francis 2000]. Revere instead relies on a single topology for both purposes, enables multi-path delivery, and enforces security with the presumption of open membership. *ALMI* builds a small-scale minimum spanning tree among end hosts, and it relies on a central controller for tree management [Pendarakis *et al.* 2001]. *End System Multicast* also targets small-scale tree-structured overlay networks, but it first builds a mesh of nodes, and then constructs a shortest-path tree out of the mesh [Chu *et al.* 2000]. *Scattercast* adopts a similar approach to *End System Multicast*, while emphasizing infrastructural support and proxy-based multicast [Chawathe 2000]. *Bayeux* [Zhuang *et al.* 2001] uses *Tapestry* [Tapestry], an application-level routing protocol, to organize receivers into a distribution tree. *Overcast* focuses on optimizing network bandwidth when building its overlay distribution tree [Jannotti *et al.* 2000]. A fundamental difference between RBone and these overlay networks is that RBone is not a tree-like structure; instead, every Revere node can choose to have two or more as-disjoint-as-possible paths to receive security updates. Also, in addition to the pushing mechanism, Revere allows each node to pull missed security updates from repository servers.

In terms of building resiliency into an overlay network, Revere shares commonalities with *RON* [Andersen *et al.* 2001]. Instead of targeting another distribution service, RON inserts a new layer of resilient overlay network between the routing substrate below and network applications above, thus providing faster routing failure recovery and application-specific routing. One useful discovery from RON is that a failed router or physical link can be avoided if a message is routed through a different node on the RON overlay.

7.3.2 Multi-Path Routing

Multi-path routing is similar to Revere's multi-path message delivery [Chen *et al.* 1998], [Murthy *et al.* 1996], [Zaumen *et al.* 1998]. However, these systems are primarily meant for load balancing or congestion avoidance and do not fully consider the disjointedness between different paths. It is also difficult for these systems to address security issues (such as key distribution, replay prevention, etc.) at router level. They also face deployment problems.

7.3.3 Peer-to-Peer Computing

Peer-to-peer computing is developing rapidly and gaining prominence as an infrastructural service. Broadly speaking, the relationship between Revere nodes is also peer-to-peer, and results from peer-to-peer research can be leveraged to improve the Revere overlay network.

7.3.4 Geographic Routing

Geographic routing sends messages to participating machines located in close physical proximity to destination points [Navas *et al.* 1997]. Primarily used to support mobile computing, geographic routing protocols may prove a useful component of the Revere system.

7.4 Security

Much research has been done on securing communication channels or strengthening networking elements. While Revere addresses its own security issues, as we discussed in Chapter 5, Security, Revere could also leverage such research. For instance, research has been performed on intrusion detection [Denning 1986], [Lunt 1988], [Snapp *et al.* 1991], [Kim *et al.* 1994], [Crosbie *et al.* 1995]. Methods that defend networks or cooperating distributed systems against intrusion, especially when all members are peers, are

particularly relevant [Mukherjee *et al.* 1994], [White *et al.* 1996], [Zerkle *et al.* 1996]. Revere is not competitive with these efforts; rather, it will be a user of existing intrusion detection techniques. We expect that the practical experience of applying these techniques to a new problem will uncover new possibilities and requirements that will call for further study.

7.5 Conclusions

In this section, we described works related to Revere. As we can see, both general-purpose distribution services and special-purpose services fail to meet the challenges of rapid, widespread and secure delivery of security updates. Information delivery structures, such as various overlay networks, while aim to provide general-purpose information delivery, do not fully address the special need of security update dissemination. Being aware of those related works, Revere provides a solution for disseminating security updates quickly, securely and resiliently. It is also able to leverage existing research results, such as security enforcement.

CHAPTER 8

Future Work

This dissertation has presented key techniques that enable the dissemination of security updates. It also provides a platform for further research on open issues. In the following two sections, we will first briefly revisit the open technical issues that have been discussed in previous chapters, and then ask questions from a broader view of the Revere system.

8.1 Open Issues Discussed in Previous Chapters

The following is a list of open issues that were more extensively discussed in prior chapters. Questions that are relevant to each item are presented as examples of issues that need to be addressed.

- **Adaptive redundancy**. How should a Revere node adjust its redundancy degree for receiving security updates? Would two different delivery paths be enough, for example? (See Section 4.6 for more details.)
- **Security update integrity protection other than using digital signature**. While digital signature based on public key cryptography has been widely used and is also employed in Revere, could other integrity protection techniques under study benefit Revere better? (See Section 4.6 for more details.)

- **Repository server selection**. Among many repository servers, which ones should a node choose to query for missing security updates? (See Section 4.6 for more details.)
- **Secure dissemination process monitoring**. How should Revere securely monitor the dissemination process in real time? How should every individual node provide feedback on its receipt of security updates? (See Section 5.5.1 for more details.)
- **RBone intrusion detection and reaction**. How should Revere detect intrusions? If a node is detected as corrupted, how can it report the problem? (See Section 5.5.2 for more details.)
- **Denial-of-service attack prevention**. Can a dissemination center be flooded and paralyzed? Can malicious nodes lock benign nodes out from receiving security updates? (See Section 5.5.3 for more details.)
- **Overloading approach improvement**. How to speed up overloading-based measurement while still collecting realistic results? To what degree can the approach be generalized? (See Section 6.5.1 for more details.)
- **Revere performance understanding at larger scale**. It is prohibitive to understand a system at very large scale, but it has always been desirable to achieve this. How can one deduce or extrapolates performance of a system from smaller-scale results? (See Section 6.5.2 for more details.)
- **Measurement results applied to the real world**. The real world is always more complex than any measurement setup. Will the results from

the latter be applicable to the real world environment? (See Section 6.5.3 for more details.)

8.2 Think More Beyond Today

Certainly more work is required on Revere to refine the general technical approach and demonstrate its possibilities. The feasibility of a delivery system of such scale and speed raises a number of serious questions:

1. **Would such a system on such a scale be valuable?** If so, then the task of deploying Revere agents on a large number of Internet nodes would probably require the public-service cooperation of major software distributors, Internet service providers, etc. This also includes testing Revere on different platforms, experimenting with its interaction with other applications, and studying wide deployment issues (some may not be technical).

2. **Can one depend on its safety?** After all, the lure of such a target to the attacker can hardly be overestimated. One could argue that addressing such a large portion of the Internet is inherently dangerous, just as one could argue that constructing a very tall building invites disaster. However, there can be variations on the delivery service. One could have several dozen, or even a few hundred Revere networks deployed, so that a successful attack on one does not immediately affect nodes that are not part of that compromised system.

   How feasible is it to validate the security of Revere? A formal method, or another approach? Can Revere automate/improve the discovery of new security problems? Recall that in Section 5.2.4.1, we discussed the detection of dissemination center impersonation. This is just a starting point.

3. **What is the quality of a delivery path at the physical level?** This question warrants study if we cannot assume that routers are fully trustworthy. In particular, there is no guarantee that if two delivery paths are disjoint at the application level, they will also be disjoint at the physical level. If not, how much overlap will there be, and is it possible that many Revere links multiplex over a specific physical link?

4. **Can a path vector carry richer information?** If so, a node would then have more information from which to select parents or adjust its own position in an RBone. But what would such information be? We know a path vector currently describes the latency of a path and the ordered list of nodes on the path, but how about the trustworthiness of a path, for example? A closely related question is, how to define the trustworthiness of a path?

5. **What will be done with the updates once delivered?** In some cases, the answer is simple and obvious, such as installing new virus signatures into a virus detection database. In other cases, there are greater challenges. For example, could the system be used to install security fixes as fast as the attacks are made? Few system administrators today are eager to accept automated patch installation, because they lack confidence that patches will work properly in their systems. If Revere were in place, safe automated patch installation would become more appealing, but Revere itself does nothing to make automated patch installation more reliable.

   Lessons can be learned from previous incidents. [Fisher 2002] reports that with at least four vaguely defined patch installation mechanisms, Microsoft's Windows Update caused the automated scanning service to mismanage patches. In one extreme case, a patch for a customer actually removed a previous hot fix, causing that machine to be infected by the Nimda virus. Worse, updates from a vendor could even conflict

with software already installed; for example, in October 2000, Symantec's new update toward a customer conflicted with its firewall, causing the firewall to shut down and leaving the affected system open to exploit [Lemos 2000].

6. **Can Revere be easily deployed in the real world**? What are those deciding factors that affect a widespread deployment of Revere? Are they mostly societal or psychological issues? For example, would another CodeRed worm trigger a fast, wide deployment of Revere? Further, how transparent should Revere be to a user? Are there a few default Revere node configurations that work well for most users? If so, how do we discover those configurations? Clearly, answers to items 1, 2, and 5 above are also critical to this issue.

7. **Can Revere be easily ported to a wireless world**? This includes the situation where *every* node is wireless and the situation where the core of a network is wired but the edge is wireless. If Revere is not readily portable here, what would those new constraints be and what redesign of Revere should be made? As we can see, when individual nodes become more mobile, the delivery paths for every Revere node will become more volatile. Meanwhile, is there something in the wireless environment that is actually more useful? For instance, will node location information as reported from GPS be critical in determining multiple physically disjoint delivery paths?

8. **Last, can an RBone be theoretically analyzed?** [Singh 1995] proposes a way to evaluate the global reliability of a communication network. Unfortunately, his method requires knowledge of the global topology of the network. In Revere, no node has such knowledge. Is there a distributed version of the algorithm where every node only has partial knowledge of the whole system?

CHAPTER 9

Conclusions

This work demonstrates that fast, secure and resilient delivery of a modest amount of information through a very large-scale network is feasible, without employing huge server farms. To summarize the work, in this chapter we will recapitulate the problem Revere tries to solve, summarize the solution Revere provides, and outline Revere's contributions. Broad lessons learned from this work will also be presented.

9.1 Summary of the Problem

Threats such as viruses, worms, or Trojan horses are able to propagate throughout the network quickly. However, potential victims do not have up-to-date knowledge of new threats, and are therefore susceptible to those threats. This indicates that any defense against threats must react at the same speed as the threat (if not faster), and that a strong need exists for a service that disseminates security updates in a fast, efficient manner. Unfortunately, despite these indications, an effective dissemination service has not existed in the past.

Although it is usually quick and easy to determine the solution to a new threat, notifying an Internet-scale network of the solution—or disseminating security updates at Internet-scale—is challenging. Such a system must outpace the spread of threats, address the complexities in a large-scale environment, ensure the delivery toward a large percentage of recipients (if not all), and secure the system itself.

Simple transmission techniques can hardly meet those requirements. Unicast requires a dissemination center to send updates toward each individual recipient, one by one; here, leaving aside the lack of bullet-proof protection of the delivery process, the sequential delivery of updates is not efficient, and the dynamic management of recipient records is not scalable. Broadcast allows all nodes in a subnet to be reached simultaneously, but the number of subnets over an Internet-scale network is still large. IP multicast builds a tree structure to deliver information, allowing a trusted source to reach all nodes connected to the tree during dissemination; however, IP multicast lacks sufficient resiliency due to its use of a tree structure. It often fails to reach nodes that have no stable connection. It has also been difficult to deploy.

Application-layer protocols and services, as seen today, can hardly be tailored or extended to fulfill the requirements, either. None of them provides an adequately resilient network to address man-in-the-middle delivery threats. Also, many of them require recipients to pull information from a source. To keep up to date, every recipient has to probe the source frequently (unless it is able to predict the availability of information in a timely fashion), potentially incurring a prohibitive bandwidth cost and processing overhead. Further, because these application-layer protocols and services have not been designed for the purpose of delivering critical updates, security concerns have been less addressed; for example, if an attacker steals the key of a dissemination source, it then can send forged information under the identity of the source.

9.2 The Revere Solution

Revere employs a dual mechanism for security update delivery: push and pull. Push allows a dissemination center to broadcast updates to all connected nodes once new updates are available; pull allows a node to catch up with missed security updates.

Push is performed through an application-layer Revere overlay network. Without relying on huge, powerful server farms, Revere adopts a non-centralized approach to build self-organized resilient overlay networks, on which every node is a recipient (except that the root is the dissemination center or a repository point), and every link is a unicast connection between a parent node and a child node. During a push session, a dissemination center only needs to forward updates to a very small number of nodes, each of which then further forwards the updates to its own children, a recursively repeatable procedure.

While the push model allows immediate delivery, the Revere overlay network further meets other challenges as follows:

- Resiliency is supported through redundancy. At its own discretion, every recipient can choose to have multiple as-disjoint-as-possible delivery paths, leading to multiple parents for the recipient in the Revere overlay network.
- Scalability is achieved through the distributed nature of the overlay network itself. In a Revere overlay network, each node need not know all the other nodes in the network. For instance, a dissemination center only needs to know all of its own direct children to begin dissemination, and a normal node only needs to know its parents, its children, the dissemination center, and a small amount of information regarding its delivery paths.
- The complexities in a large-scale environment are handled by designing the Revere overlay network to be self-organized and without a central control. A Revere overlay network can be fairly dynamic, and the large scale of the overlay further complicates the problem: nodes may come and

go, links could go up and down, and the whole overlay can hardly stay in a steady state. Via self-organization, new nodes can join through the three-way-handshake, existing nodes can detect problems and reposition themselves. The redundancy built in the overlay network also provides a cushion period while a node is adjusting its position—it still can receive security updates during the transient period unless all its paths are broken.

- The overlay network operates at the application layer. New functionalities can be easily added, and different configurations set up without difficulty. Implemented in Java, Revere can be readily installed and started. Once a node begins to run Revere, it will automatically join a relevant Revere overlay network and become a Revere node. Leaving Revere is equally easy. In light of these features, deployment of Revere becomes easy.

Pull is performed through contacting repository servers. Repository servers, either statically configured or dynamically elected, allow a reconnected node to pull missed security updates. To obtain higher certitude when pulling missed security updates, the node can independently contact multiple repository servers. The pull mechanism also provides a means for a node that is connected all the time to determine if all its parents have been compromised and are blocking him from receiving security updates.

Both push and pull processes are secured. Revere uses public key cryptography to protect security updates. Every security update must be signed by its dissemination center, and every node must verify the authenticity and integrity of an update (whether the update is pulled from a repository server or pushed from a parent node). Moreover, once a dissemination center detects that its private key is stolen, it will immediately initiate the key invalidation process. Upon the receipt of a (verified) key invalidation

message, a node will discard the current public key in use and switch to the next one. There is an important property regarding the key invalidation message: whether injected by a trusted dissemination center or an attacker, it will not degrade the security of a Revere overlay network.

The Revere overlay network is also secured to provide a sound delivery structure. While most Revere nodes can be assumed cooperative, Revere is designed to deal with the corruption of some nodes. Knowing that all nodes in an Internet-scale overlay network cannot enforce a uniform security scheme, Revere supports pluggable security boxes to easily enforce new security schemes. For two nodes that enforce different sets of security schemes, the peer-to-peer security scheme negotiation will allow them to communicate, and do so securely, by following the security preference of each other.

Key performance results of a prototype, measured using the large-scale-oriented overloading approach, suggest that Revere can deliver security updates at the required scale, speed and resiliency for a reasonable cost. For instance, it takes less than 1 second to reach all nodes in a 3000-node Revere overlay network, and just a bit over 1 second to reach 93% of working nodes even if 15% of all nodes are broken. The cost for joining such a network is also lightweight, with a less-than-two-second join latency and about 20 kilobytes of bandwidth cost.

9.3 Contributions of the Dissertation

The thesis of this research is that without relying on huge, powerful server farms, it is still feasible to deliver a modest amount of information in an Internet-scale network quickly, securely and resiliently. Solutions based on other distribution services or delivery structures are frequently insufficient when security and resiliency requirements become critical. Instead, this research provides a sound solution for such delivery.

Revere introduces a dual mechanism for delivering critical information. With push and pull combined, not only can information be broadcast to recipients once it becomes available, but it can also be made available for recipients to query at any time.

Revere builds a self-organized resilient overlay network for large-scale delivery. In such an overlay network, every node can both receive information and forward information, acting as both a beneficiary and a benefactor. Different from other overlay networks, a Revere overlay network allows a node to select multiple least-overlapping delivery paths, and achieve best resiliency using a path vector concept. Also different from many other overlay networks, Revere supports open membership, instead of enforcing closed membership and ensuring that every member is trusted. Because of such differences, management of the Revere overlay network becomes more challenging; Revere has built a self-organizing capability into its overlay networks to cope with complexities in a dynamic large-scale environment.

Revere protects both the delivery procedure and the delivery structure. For the former, a digital signature allows a node to verify the authenticity and integrity of security updates, redundancy in both push and pull allows a node to gain higher certitude in receiving security updates, and the key invalidation mechanism allows a node to discard a corrupted public key and switch to a new one. For the latter, the implementation of pluggable security boxes allows a node to flexibly enforce different security schemes and policies, the peer-to-peer security scheme negotiation enables a node to enforce its own specific security schemes when communicating with other nodes, and the discretionary security enforcement at individual nodes makes the whole delivery structure robust.

Revere also provides a deployable solution. Revere runs at application level, and it does not need any changes to underlying operating systems or network infrastructures.

Any node can automatically join a Revere network by running Revere software (and certainly it is also easy for a node to leave the network). Without demanding a large amount of node or networking resources, Revere supports lightweight core functionalities. Revere is modular and extensible; for example, Revere is easily extensible for adding new security schemes, readily configurable for replacing the default policy for adopting a new child or parent, and fairly adaptive for tailoring to locally available transmission mechanisms.

9.4 Broad Lessons

Without using large server farms, Revere becomes powerful by aggregating resources among recipients: every recipient, as beneficiary of the service, can also serve as a benefactor. This liberates a dissemination source from being solely responsible for reaching all nodes in the system. For this reason, the Revere model is not a purely client-server or publisher-subscriber model, since a normal node also serves others. It underpins a simple but profound fact: a node can obtain information goods not only from the original servers, but also from its peers. Instead of being a negative factor, larger scale in this context actually means more sources from which to gain information. Peer-to-peer computing, for instance, echoes this principle in distributing data among all the peering nodes and benefits all the peering nodes.

Revere is also an example of a design that crosses three fields: distributed systems, security and networking. Composed of mostly cooperative elements but also some uncooperative ones, its task of synchronizing all elements to have the same information (security updates) across the network in the face of security challenges is a typical mission for such a field-crossing design. These field-crossing designs face challenges similar to those that Revere faces (efficiency, scalability, security, resiliency, etc.), where

some elements may not be trusted (subject to corruption), reliable (prone to error), or always available (subject to failure). The methodologies adopted by Revere to be robust to such problems are also applicable to those similar designs.

Revere is also a service that delivers information at application level. It demonstrates that an application-level Revere-like service is feasible and can be made effective without changing underlying hardware, operating systems, or network infrastructures. Further, Revere shows an interesting phenomenon in its incremental deployment: not only can Revere be easily deployed (every node can just run Revere software to become a Revere node), but Revere also tends to show more attractive benefits to potential participants as more nodes exist in the system and form a larger information pool.

9.5 Final Comments

Since today's attackers already distribute malicious functions rapidly, an even faster notification system is required. Although additional work is necessary in order to ensure a very high certainty of safety and to verify feasibility, Revere offers encouragement that such a system is possible.

Revere also has a broader applicability for delivering other information rather than just security updates. For instance, when every node in a Revere overlay network chooses to only have a single delivery path, the Revere overlay network effectively becomes a tree structure, and Revere is then equivalent to a system that supports host-level multicast.

Likewise, the overloading technique used for measuring Revere has a broader use. Other distributed systems may find this large-scale-oriented approach helpful. For example, this research has developed techniques to adjust those results that are affected

by the degree of overloading. Our experiences related to this overloading technique can lend insights for testing other systems.

Broader use of redundancy is also very appealing. Revere demonstrates that proper use of redundancy will greatly improve system resiliency and information readiness. Given the abundance of many kinds of components in an Internet-scale distributed system, chances are that Revere-like redundancy, or redundancy at much greater scale, will prove very advantageous.

TRADEMARKS

LINUX is a trademark of Linus Torvalds. RED HAT is a trademark of Red Hat Software, Inc. JAVA is a trademark of Sun Microsystems, Inc. AMD is a trademark of Advanced Micro Devices, Inc. ETHERNET is a trademark of Xerox Corporation. CERT is a trademark of Carnegie Mellon University. ICSA is a trademark of ICSA, Inc. BBC is a trademark of the British Broadcasting Corporation. VERISIGN is a trademark licensed to VeriSign, Inc. MICROSOFT and WINDOWS are trademarks of Microsoft Corporation. KERBEROS is a trademark of the Massachusetts Institute of Technology. RSA is a trademark of RSA Security Inc. iBEAM Broadcasting is a trademark of iBEAM Broadcasting Corporation. CacheWare is a trademark of CacheWare, Inc. Digital Island is a trademark of Digital Island, Inc. epicRealm is a trademark of epicRealm Operating Inc. PointCast is a trademark of PointCast, Inc. Xerox is a trademark of Xerox Corporation. BackWeb is a trademark of BackWeb Technologies. iFusion is a trademark of iFusion, LLC. McAfee is a trademark of Network Associates, Inc. Symantec is a trademark of Symantec Corporation. Ricochet is a trademark of Metricom, Inc.

REFERENCES

[Alonso *et al.* 1989] R. Alonso, D. Barber, and S. Abad. "A File Storage Implementation for Very Large Distributed Systems," *IEEE Workshop on Workstation Operating Systems*, September 1989.

[Alteon] *White Paper*. "Enhancing Web User Experience With Global Server Load Balancing," Alteon Networks. Available at http://www.alteon.com/products/white_papers/GSLB/index.html.

[Andersen *et al.* 2001] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. "Resilient Overlay Networks," *SOSP* 2001.

[Badrinath *et al.* 1992] B. Badrinath and T. Imielinski. "Replication and Mobility," *Proceedings of the 2nd IEEE Workshop on Mobility of Replicated Data*, November 1992.

[Bannister *et al.* 1997] J. Bannister, R. Lindell, C. DeMatteis, M. O'Brien, J. Stepanek, M. Campbell, and F. Bauer. "Deploying Internet Services Over a Direct Broadcast Satellite Link: Challenges and Opportunities in the Global Broadcast Service," *Proceedings of MILCOM 97*, 1997.

[Bauer *et al.* 1992] M. A. Bauer and T. Wang. "Strategies for Distributed Search," *Proceedings of the 1992 ACM Computer Science Conference on Communications*, 1992.

[BBC News 2000] BBC News. "Police close in on Love Bug culprit," Saturday, 6 May 2000. http://news.bbc.co.uk/hi/english/sci/tech/newsid_738000/738537.stm.

[Bertsekas *et al.* 1992] D. Bertsekas and R. Gallagher. *Data Networks*, Prentice Hall, 1992.

[Birman 1985] K. Birman. "Replication and Availability in the ISIS System," *Proceedings of the ACM Symposium on Operating System Principles*, December 1985.

[Birman *et al.* 1991] K. Birman, A. Schiper, and P. Stephenson. "Lightweight Causal and Atomic Group Multicast," *ACM Transactions on Computer Systems*, Vol. 9, No. 3, 1991.

[Cabrera *et al.* 2001] L. Cabrera, M. Jones, and M. Theimer. "Herald: Achieving a Global Event Notification Service," *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau, Germany, May 2001.

[CAIDA 2001] CAIDA. "The Spread of the Code-Red Worm (CRv2)," http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml, and http://worm-security-survey.caida.org/.

[Calvert *et al.* 1997] K. Calvert, M. Doar, and E. Zegura. "Modeling Internet Topology," *IEEE Comm. Magazine* 35, 6 June 1997.

[Case *et al.* 1990] J. Case, M. Fedor, M. Schoffstall and J. Davin. "A Simple Network Management Protocol (SNMP)," RFC 1157, May 1990.

[Castro *et al.* 1999] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance," *Proceedings of the Third Symposium on Operating Systems Design and Implementation,* New Orleans, February 1999.

[CERT 2000] Computer Emergency Response Team. "CERT(R) Advisory CA-2000-01 Denial-of-Service Developments," http://www. cert.org/advisories/CA-2000-01.html, January 2000.

[CERT 2001:1] Computer Emergency Response Team. "CERT(R) Advisory CA-2001-19 `Code Red' Worm Exploiting Buffer Overflow in IIS Indexing Service DLL," http://www.cert.org/advisories/CA-2001-19.html, July 19, 2001.

[CERT 2001:2] Computer Emergency Response Team. "CERT(R) Advisory CA-2001-23 Continued Threat of the `Code Red' Worm," http://www.cert.org/advisories/CA-2001-23.html, July 26, 2001.

[Chang *et al.* 1984] J. M. Chang and N. F. Maxemchuck. "Reliable Broadcast Protocols," *ACM Transactions on Computing Systems,* August 1984.

[Chawathe 2000] Y. Chawathe. "Scattercast: an Architecture for Internet Broadcast Distribution as an Infrastructure Service," Ph.D. thesis, Fall 2000.

[Chen *et al.* 1998] J. Chen, P. Druschel, and D. Subramanian. "An Efficient Multipath Forwarding Method," *IEEE INFOCOM* 1998.

[Cheung *et al.* 1997] S. Cheung and K. Levitt. "Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection," *Proceedings of New Security Paradigms Workshop*, Langdale, Cumbria, UK, 1997.

[Chu *et al.* 2000] Y. Chu, S. Rao, and H. Zhang. "A Case for End System Multicast," *Proceedings of ACM Sigmetrics,* June 2000.

[Crosbie *et al.* 1995] M. Crosbie and G. Spafford. "Defending a Computer System Using Autonomous Agents," *Proceedings of the 18th National Information Systems Security Conference,* 1995.

[Danzig *et al.* 1994] P. Danzig, K. Obraczka, D. DeLucia, and N. Alam. "Massively Replicating Services in Autonomously Managed Wide-Area Internetworks," *USC Computer Science Department Technical Report 93-541*, January 1994.

[Deering 1989] S. Deering. "Host Extension for IP Multicasting," RFC-1112, August 1989.

[Demers *et al.* 1987] A. Demers et al. "Epidemic Algorithms for Replicated Database Management," *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, 1987.

[Demers *et al.* 1994] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. "The Bayou Architecture: Support for Data Sharing Among Mobile Users," *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, December 1994.

[Denning 1986] D. Denning. "An Intrusion Detection Model," *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, 1986.

[Dijkstra *et al.* 1959] E. Dijkstra. "A Note On Two Problems in Connexion with Graphs," *Numerische Mathematik*, 1:269-271, 1959.

[Doar 1996] M. B. Doar. "A Better Model for Generating Test Networks," *Proceedings of Global Internet*, November 1996.

[Felix] Project Felix: Independent Monitoring for Network Survivability. Available at http://govt.argreenhouse.com/felix/.

[Fisher 2002] D. Fisher. "Security Tool Leaves Holes," *eWeek—the Enterprise Newsweekly*, Vol. 19, No. 16, April 22, 2002.

[Floyd *et al.* 2001] S. Floyd and V. Paxson. "Difficulties in Simulating the Internet," *IEEE/ACM Transactions on Networking*, February, 2001.

[Floyd *et al.* 1995] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *Proceedings of the ACM SIGCOMM 95*, 1995.

[Francis 2000] P. Francis. Yoid: Your Own Internet Distribution. Available at http://www.aciri.org/yoid, April 2000.

[Francis *et al.* 2001] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. "IDMaps: a Global Internet Host Distance Estimation Service," *IEEE/ACM Transactions on Networking*, 9(5): 525-540, October 2001.

[Frank 2000] D. Frank. "One if by phone, two if by fax," *Federal Computer Week*, September 2000.

[Fyodor 2001] Fyodor. nmap ("Network Mapper"). http://www.insecure.org/nmap. 2001.

[Garcia-Lunes-Aceves 1993] J. J. Garcia-Lunes-Aceves. "Loop-Free Routing Using Diffusing Computations," *IEEE Transactions on Networking*, Vol. 1, No. 1, February 1993.

[Goel 1996] A. Goel. "View Consistency for Optimistic Replication," *UCLA Technical Report CSD-960011*, February 1996.

[Goldreich *et al.* 1992] O. Goldreich and D. Sneh. "On the Complexity of Global Computation in the Presence of Link Failures: The Case of Uni-Directional Faults," *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing*, 1992.

[Golding *et al.* 1993] R. Golding and D. Long. "Modeling Replica Divergence in a Weak-Consistency Protocol for Global-Scale Distributed Data Bases," *UC Santa Cruz Computer Science Department Technical Report UCSC-CRL-93-03*, 1993.

[Gray *et al.* 1996] J. Gray, P. Helland, P. O'Neil, and D. Shasha. "The Dangers of Replication and a Solution," *Proceedings of the ACM SIGMOD Conference*, June 1996.

[Gruber *et al.* 1999] R. Gruber, B. Krishnamurthy, E. Panagos. "The Architecture of the READY Event Notification Service," *Proceedings of 19th IEEE International Conference on Distributed Computing Systems*, May 1999.

[Guy *et al.* 1990] R. Guy, J. Heidemann, W. Mak, T. Page, G. Popek, and D. Rothmeier. "Implementation of the Ficus Replicated File System," *Proceedings of the 1990 Usenix Conference*, June 1990.

[Hisgen *et al.* 1990] A. Hisgen, A. Birrell, C. Jerian, T. Mann, M. Schroeder, and G. Swart. "Granularity and Semantic Level of Replication in the Echo Distributed File System," *Proceedings of the IEEE Workshop on Management of Replicated Data*, November 1990.

[Olavsrud 2001] T. Olavsrud. "Fraudulent Digital Certificates Issued in Microsoft's Name," *Internet News*, March 22, 2001. http://www.internetnews.com/dev-news/article/0,,10_721571,00.html.

[Jannotti *et al.* 2000] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole Jr. "Overcast: Reliable Multicasting with an Overlay Network," *Proceedings of the Fourth Symposium on Operating System Design and Implementation*, October 2000.

[Jin *et al.* 2000] C. Jin, Q. Chen, and S. Jamin. "Inet: Internet Topology Generator," University of Michigan Technical Report CSE-TR-433-00, 2000.

[Jou *et al.* 1997] Y. Jou, F. Gong, C. Sargor, S. Wu, R. Cleaveland. "Architecture Design of a Scalable Intrusion Detection System for the Emerging Network Infrastructure," MCNC Technical Report CDRL A005, April 1997. Available at http:// shang.csc.ncsu.edu//papers/jinaoArch.ps.gz.

[Kaashoek *et al.* 1989] M. F. Kaashoek, et al. "An Efficient Reliable Broadcast Protocol," *ACM Operating Systems Review*, Vol. 23, No. 4, October 1989.

[Kashima 1995] H. Kashima. "Searching Internet Resources Using IP Multicast," *INET '95*, August 1995.

[Kephart *et al.* 1997] J. O. Kephart, G. B. Sorkin, D. M. Chess, and S. R. White. "Fighting Computer Viruses," *Scientific American*, November 1997.

[Kim *et al.* 1994] G. Kim and E. Spafford. "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool," *Purdue University Computer Science Department Technical Report CSD-TR-94-019*, 1994.

[Kistler *et al.* 1991] J. Kistler, and M. Satyanarayanan. "Transparent Disconnected Operation for Fault-Tolerance," *ACM Operating Systems Review*, Vol. 25, No. 1, January 1991.

[Krishnamurthy *et al.* 1995] B. Krishnamurthy and D. Rosenblum. "Yeast: a General Purpose Event-Action System," *IEEE Transactions on Software Engineering*, 21(10), October 1995.

[Kuenning *et al.* 1997] G. Kuenning and G. Popek. "Automated Hoarding for Mobile Computers," *Proceeding of the 16th ACM Symposium on Operating Systems Principles* (SOSP-16), October 5-8, 1997.

[Lemos 2000] R. Lemos. "Bugs at Internet Speed?" *ZDNet*, November 2000. http://www.zdnet.com/zdnn/stories/news/0,4586,250123,00.html.

[Levine *et al.* 1996] B. Levine, D. Lavo, and J. J. GarciaLuna-Aceves. "The Case for Reliable Concurrent Multicasting Using Shared Ack Trees," *Proceedings of the ACM Multimedia Conference*, November 1996.

[Li *et al.* 1999] J. Li, P. Reiher, and G. Popek. "Securing Information Transmission by Redundancy," *Proceedings of New Security Paradigms Workshop*, ACM SIGSAC, September 1999.

[Li *et al.* 2002] J. Li, M. Yarvis, and P. Reiher. "Securing Distributed Adaptation," *Computer Networks, Special Issue on Programmable Networks*, 38(3): 347-371, Elsevier Science, 2002.

[Liebeherr *et al.* 1997] J. Liebeherr and B. Sethi. "A Scalable Control Topology for Multicast Communications," *Personal Communications*, Polytechnic University of New York, Brooklyn, 1997.

[Lin *et al.* 1996] J . C. Lin and S. Paul. "RMTP: A Reliable Multicast Transport Protocol," *Proceedings of IEEE INFOCOM*, pp. 1414-1425, March 1996.

[Liskov *et al.* 1991] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams. "Replication in the Harp File System," *Proceedings of the ACM Symposium on Operating System Principles*, October 1991.

[Lunt 1988] T. Lunt. "Automated Audit Trail Analysis and Intrusion Detection: A Survey," *Proceedings of the 11th National Computer Security Conference*, October 1988.

[Macedonia *et al.* 1997] M. Macedonia and D. Brutzman. "MBONE: The Multicast BackBONE," http://www.mice.cs.ucl.ac.uk/- mice/mbone review.html, 1997.

[Malan *et al.* 1997] G. R. Malan, F. Jahanian, and S. Subramanian. "Salamander: A Push-based Distribution Substrate for Internet Applications," *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[McAfee Rumor] McAfee ASaP's Rumor Technology. http://www.mcafeeasap.com/ content/virusscan_asap/rumor.asp.

[McAfee OilChange] McAfee OilChange Online (previously CyberMedia OilChange). http://www.mcafee.com/myapps/oco/default.asp?

[McDermott 1997] J. McDermott. "Replication Does Survive Information Warfare Attacks," *Proc. of the 11th Annual Working Conference on Database Security*, August 1997, pp. 186-198.

[Medina *et al.* 2000] A. Medina, I. Matta, and J. Byers. "On the Origin of Power Laws in Internet Topologies," *ACM Computer Communication Review*, 30(2), April 2000.

[Mills 1991] D. Mills. "Internet Time Synchronization: The Network Time Protocol," *IEEE Transactions on Communications*, Vol. 39, No. 10, October 1991.

[Moser *et al.* 1997] L. E. Moser and P. M. Melliar-Smith. "Secure Multicast Protocols for Group Communications," Technical Report, Department of Electrical and Computer Engineering, University of California at Santa Barbara, 1997.

[Moses *et al.* 1989] Y. Moses and G. Roth. "On Reliable Message Diffusion," *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, 1989.

[Moskowitz *et al.* 1997] I. Moskowitz and M. Kang. "An Insecurity Flow Model," *Proc. of New Security Paradigms Workshop*, Cumbria, UK, September 1997.

[Mukherjee *et al.* 1994] B. Mukherjee, L. Heberlein, and K. Levitt. "Network Intrusion Detection," *IEEE Network*, May/June 1994.

[Murthy *et al.* 1996] S. Murthy and J. J. Garcia-Luna-Aceves. "Congestion-Oriented Shortest Multipath Routing," *IEEE INFOCOM*, 1996.

[Navas *et al.* 1997] J. Navas and T. Imielinski. "Geographic Addressing and Routing," *Proceedings of the Third ACM/IEEE International Conference on Mobile Computing*, 1997.

[Page *et al.* 1998] T. Page, R. Guy, J. Heidemann, D. Ratner, P. Reiher, A. Goel, G. Kuenning, and G. Popek. "Perspectives on Optimistically Replicated Peer-to-Peer Filing," *Software Practice and Experience*, 1998.

[Patrizio 2000] A. Patrizio. "Content-Delivery Network Services Vary Greatly," *INFORMATIONWEEK.com*, December 2000. http://www.informationweek.com/815/cdnvendors.htm.

[Patterson *et al.* 1989] D. Patterson, G. Gibson, and R. Katz. "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of IEEE COMPCON*, Spring 1989.

[Paxson 1996] V. Paxson. "End-to-End Routing Behavior in the Internet," *Proceedings of ACM Sigcomm, 1996.*

[Pelc 1996] A. Pelc. "Fault-Tolerant Broadcasting and Gossiping in Communication Networks," *Networks*, Vol. 28 (1996), pp. 143-156.

[Pendarakis *et al.* 2001] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. "ALMI: an Application Level Multicast Infrastructure," in *Proceedings of the 3rd Usenix Symposium on Internet Technologies & Systems (USITS)*, March 2001.

[Pingali *et al.* 1994] S. Pingali, D. Towsley, and J. F. Kurose. "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols," *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, issued as Performance Evaluation Review, Vol. 22, No.1, pp. 221-330.

[PCCIP 1997] The President's Commission on Critical Infrastructure Protection. Critical Foundations: Protecting America's Infrastructure, October 1997.

[Postel 1981] J. Postel. "Transmission Control Protocol," RFC 0793, September 1981.

[Quarterman *et al.*] J. S. Quarterman and P. H. Salus. "How the Internet Works," *Matrix.net*. http://www.mids.org/works.html.

[Rabin 1989] M. Rabin. "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance," *JACM* 36(2) (1989), pp. 335-348.

[Ratnasamy *et al.* 2001] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. "A Scalable Content-Addressable Network," *ACM Sigcomm 2001*, August 2001.

[Ratner 1998] D. Ratner. "Roam: A Scalable Replication System For Mobile and Distributed Computing," UCLA CSD Ph.D. dissertation, January 1998.

[Reiher *et al.* 1993] P. Reiher, T. Page, S. Crocker, J. Cook, and G. Popek. "Truffles - A Secure Service for Widespread File Sharing," *Proceedings of the Privacy and Security Research Group Workshop on Network and Distributed System Security*, February 1993.

[Reiher *et al.* 1996] P. Reiher, G. Popek, M. Gunter, J. Salomone, and D. Ratner. "Peer-to-Peer Reconciliation-Based Replication for Mobile Computers," *Proceedings of the ACM/ECOOP Workshop on Mobility and Replication*, July 1996.

[RFC919] Request for Comments 919 – Broadcasting Internet Datagrams.

[RFC922] Request for Comments 922 - Broadcasting Internet Datagrams in the Presence of Subnets.

[RFC947] Request for Comments 947 - Multinetwork Broadcasting Within the Internet.

[RFC1825] Request for Comments 1825 - Security Architecture for the Internet Protocol.

[Rosenstein *et al.* 1997] A. Rosenstein, J. Li, and S. Tong. "MASH: the Multicasting Archie Server Hierarchy," *Computer Communication Review*, Vol.27, No.3, ACM SIGCOMM, July 1997, pp. 5-13.

[Satyanarayanan *et al.* 1990] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990.

[S-BGP] Secure BGP Project (S-BGP). Available at http://www.net-tech.bbn.com/sbgp/sbgp-index.html.

[Snapp *et al.* 1991] S. Snapp, J. Brentano, G. Dias, T. Goan, T. Heberlein, C. Ho, K. Levitt, B. Mukherjee, S. Smaha, T. Grance, D. Teal, and D. Mansur. "DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and an Early Prototype," *Proceedings of the 14th National Computer Security Conference*, 1991.

[Singh 1995] B. Singh. "A Global Reliability Evaluation: Cutset Approach," *IETE Technical Review*, Vol 12, No 4, July-August 1995, pp. 275-278.

[Singhal *et al.* 1994] M. Singhal and N. Shivaratri. *Advanced Concepts in Operating Systems, Distributed, Database, and Multiprocessor Operating Systems*, McGraw-Hill, Inc., 1994

[Stoica *et al.* 2001] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. "Chord: a Scalable Peer-to-Peer Lookup Service for Internet Applications," *ACM Sigcomm 2001*, August 2001.

[Symantec TuneUp] Quarterdeck Tuneup (merged with Symantec Corp.). Available at http://www.tuneup.com.

[Tapestry] Tapestry: Fault-resilient Wide-area Location and Routing. http://www.cs.berkeley.edu/~ravenben/tapestry.

[Venkateswaran *et al.* 1997] R. Venkateswaran, C. Raghavendra, X. Chen, and V. Kumar. "DMRP: A Distributed Multicast Routing Protocol for ATM Networks," *Proceedings of the ATM 97 Workshop*, 1997.

[Wang *et al.* 1997] F. Wang, B. Vetter, and S. Wu. "Secure Routing Protocols: Theory and Practice," May 1997. Available at http://shang.csc.ncsu.edu//papers/CCR-SecureRP2.ps.gz.

[Whetten *et al.* 1995] B. Whetten, S. Kaplan, and T. Montgomery. "A High Performance Totally Ordered Multicast Protocol," *Proceedings of IEEE INFOCOM*, 1995.

[White *et al.* 1996] G. White, E. Fisch, and U. Pooch. "Cooperating Security Managers: A Peer-Based Intrusion Detection System," *IEEE Network*, January/February 1996.

[Wu *et al.* 1998] S. Wu and C. Sargor. "Deciduous: Decentralized Source Identification for Network-based Intrusions," *DARPA/ITO Next Generation Internet PI Conference*, October 1998. Available at http://shang.csc.ncsu.edu/deciduous/.

[Yavatkar *et al.* 1993] R. Yavatkar and L. Manoj. "Optimistic Strategies for Large-Scale Dissemination of Multimedia Information," *ACM Multimedia 93 Proceedings*, 1993.

[Yavatkar *et al.* 1995] R. Yavatkar, J. Griffioen, and M. Sudan. "A Reliable Dissemination Protocol for Interactive Collaborative Applications," *ACM Multimedia 95 Proceedings*, 1995.

[Zaumen *et al.* 1998] W. T. Zaumen and J.J. Garcia-Luna-Aceves, "Loop-Free Multipath Routing using Generalized Diffusing Computations," *IEEE INFOCOM* 1998.

[Zerkle *et al.* 1996] D. Zerkle and K. Levitt. "NetKuang: A Multi-host Configuration Vulnerability Checker," *Proceedings of the 6th USENIX Security Symposium*, July 1996.

[Zhuang *et al.* 2001] S. Zhuang, B. Zhao, A. Joseph, R. Kata, and J. Kubiatowicz. "Bayeux: an Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination," *Proceedings of NOSSDAV*, 2001.