Dissemination Of Security Updates:
A dissertation prospectus

Jun Li

November 18, 1998

Abstract

There are many situations where dissemination of security updates is highly needed. To disseminate security updates over Internet to large number of machines securely, quickly, adaptively and with high assurance is a very challenging work. We propose to achieve this by designing a system called *Revere*. We will analyze the way to fight against attacks, dissemination mechanisms such as high assurance through redundancy, the dissemination structure formation and management, etc. We'll also describe one strategy in study.

1 Introduction

1.1 Motivation

1.1.1 Counteracting threats as a whole

Protection of information infrastructure is inherently a distributed task. Threats as a whole must be countered, instead of just focusing on a single machine. In principle, each machine connected to network needs to be careful of possible attacks. Particularly, in a networked environment, the threats to one single machine are of concern to other machines on which same or similar attacks could also occur likely, typically through propagation and replication. With the rapid growth of network, a threat has potential to endanger more and more machines. On the other hand, it's usually the case that a threat or vulnerabilities to many machines is detected in a small number of machines first.

1.1.2 Examples

One typical example is the virus signature distribution. Currently many groups devote substantial efforts to identifying and combating new viruses soon after they are discovered. However, the information distribution of the newly detected virus is still primary. It's essentially done by each user to go to a web site to download the newly updated anti-virus software, or get the new virus signature directly from the site. This is not a scalable solution when large amount of users want to get the information of the new virus; it's also hard for the user to timely update their anti-virus software since they have to know in advance that some information about a new virus is published. Also for the anti-virus software itself, it has to be updated manually by incorporating the new signature, or to be replaced by the new version.

Another example is the kid-protector software, by which a kid is prohibited from viewing a list of web sites. However, the addresses of those sites keep on growing or changing. To make the local list of the site addresses of the kid-protector most up-to-date, an automatic and resilient dissemination of these addresses is very helpful and important.

The filter in a firewall may also like to always keep a most up-to-date list of offending characteristics, such as a list of some IP addresses. And to do this work manually is error-prone and usually not up-to-date.

1.1.3 Report of President's Commission

In [PRES97], the President's Commission on Critical Infrastructure Protection, after a wide investigation and analysis, drew a conclusion that *the quickest and most effective way to achieve a much higher level of protection from cyber threats is a strategy of cooperation and information sharing based on partnerships among the infrastructure owners and operators and appropriate government agencies* and *a real-time capability for attack warning* is a very needed mechanism *to facilitate this new relationship between government and industry.*

1.1.4 Solution is information dissemination

An obvious solution is to disseminate the information of the threats to all concerning machines in an effective, secure, and efficient fashion. We'd very much like to see a solution by which a small number of dissemination source can disseminate security updates to a large number of machines over Internet.

1.1.5 Other applications

There are also other situations that a dissemination facility of security updates is strongly needed. Cooperating intrusion detectors in a distributed environment need to exchange status and event information. System state awareness and risk assessment is dependent on timely, trustworthy security status updates. Adapting systems to tolerate or recover from attacks depends on knowing the characteristics of those attacks. These and other distributed security services have a common need to exchange security-related information among large numbers of nodes relatively quickly and with strong protections.

1.1.6 Our work

Rather than rebuilding the service for each different system that requires secure information dissemination, a shared service that can deliver security updates for all systems that need them will prove to be very useful, convenient and important. We plan to build such a service called *Revere*. Each machine who wants to receive the security updates through Revere system is named a *Revere node*. The structure over which Revere dissemination happens is named *RBone*.

1.2 Challenges

There are some challenges that a successful Revere solution must face:

1. While used for the dissemination of security updates, the Revere working mechanism itself must be robust and secure. Revere itself can be an extremely attractive target of security attack because of its purpose. If the security of Revere is broken, the machines intended to be protected by Revere will lose all of the protective powers that Revere provided. Worse than that, Revere is a system that can alter the behavior of large number of machines connected to the Internet if Revere is widely deployed, so a broken Revere may be utilized by hostile forces to corrupt large number of machines.

2. The dissemination of security updates must be very fast, adaptive, and resilient to the failure or temporary disconnection of some Revere nodes, caused by either physical or security reasons.

   A highlight here is that Revere wants to provide high availability of security updates, even under the circumstances that somebody is trying to corrupt the information in the middle. Some Revere nodes may already be corrupted and even try to corrupt other machines and break the dissemination of security updates. The dissemination of security updates by Revere has to compete with the propagation of malicious code and achieve the most possible availability of security updates.

   Moreover, the whole Revere system is dynamic. Some new machines may join and some existing machines may leave, and some may even crash or be corrupted. Some Revere nodes may be disconnected for some time. Revere needs to adapt itself to all these situations quickly through the distributed cooperation among all elements.

3. There are currently(1998) tens of millions of machines connected into Internet, and each machine is a potential node to run Revere. The scale of Internet is still growing. Thus for a widely used Revere, it's very hard for a single machine or even dozens of machines to store all the needed information, such as per-node or per-domain data structures. Some Revere nodes may only have limited communication and computing capability. Even if this is feasible by having some extremely powerful machines, to keep the stored information up-to-date is daunting. This makes a centralized management of Revere impossible. Moreover, running over a huge number of machines, the security of Revere has to be still very strong, and the performance has to be still acceptable with low overhead.

4. A good Revere solution running over heterogeneous platforms of different hardware, operating system and communication protocols ought to be easily portable and interoperable. Moreover, each Revere node may have different request on the quality of disseminating the security updates in terms of the security level and speed.

5. While facing so many challenges, Revere needs to be low overhead and light-weight while still solving these problems, because Revere is used for abnormal behavior prevention, and under normal conditions, it is purely overhead which a user only would be happy to pay as a security cost.

1.3 Thesis

We plan to investigate the feasibility of building such an automated system which can successfully disseminate security updates over Internet in a secure, fast and adaptive fashion, with high data availability assurance, to a large number of machines.

In the work we'll address the challenges proposed in subsection 1.2. We will particularly address the following several problems:

1. Formation and management of *RBone*. We'd like the RBone to be formed through automatic configuration, instead of manually setup. Also, we'd like the management of RBone to be automatically adaptive to the changes.

2. Building multipath mechanism into the RBone to achieve high assurance, so that each Revere node can have more than one path to get updated. A Revere node is expected to get security updates through two ways: security updates push originated from a dissemination source, and security updates pull initiated by a Revere node which missed some security updates. *And*

3. Security enforcement. We want to protect security updates to disseminate (by authentication and encryption, for instance), we also want to protect Revere itself against attacks from internal or external. Revere can not assume that all Revere nodes participate in a completely cooperative way, although it may have to assume that large percentage of Revere nodes are cooperative.

In doing so, four important issues to pay particular attention:

- Scalability;
- Adaptability;
- Receiver-based policy enforcement;
- Reasonable and opportunistic resource utilization;

We hope that such an automated facility can be built in a sufficiently general fashion to be applicable to a wide range of styles, such as virus signature dissemination, event and status notification in distributed intrusion detection framework, and other situations where dissemination of secure updates is needed.

We also expect to construct our system to operate with as little as possible user intervention.

Finally, Revere's goal is to protect a large percentage of Revere nodes, not necessarily to protect all nodes perfectly.

1.4 Related Work

1.4.1 Broadcasting, Flooding, and Other Dissemination Techniques

The Revere data dissemination problem has similarities to several data broadcasting services and products. Existing Internet broadcast mechanisms that work at the IP level use best-efforts delivery [RFC919], [RFC922]. They do not guarantee delivery, and they have no concern for security. Further, they are not intended to deal with the kind of scale that Revere will handle. They are meant for broadcast to subnets or small collections of subnets.

Broadcast can be achieved by a simple flooding algorithm which does not require maintaining any state or topological information [GOLD92], [BERT92], [BAUE92]. Each node that receives the broadcast message transmits on all interfaces other than the incoming one. After sufficient steps, the message will reach all the nodes. Flooding methods tend to be inefficient in their use of bandwidth, as 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.

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.

IPSEC is intended to provide general secure transmission of IP messages in the Internet [RFC1825]. IPSEC is not yet deployed, and is not specifically meant for multicast, though it is intended to have sufficient generality to support multicast. IPSEC itself has no concern for issues of secure multicast.

Many commercial products and research projects support data broadcasting [BANN97]. 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, nor handle disconnected and mobile nodes, nor authenticate messages.

Other commercial products that provide information dissemination services include BackWeb, Ifusion, InCommon, Intermind, Marimba, NETdelivery, and

Wayfarer. Many of these commercial systems require the clients to periodically poll the servers for new data, fetching it if available. This approach has performance problems at high scale, especially if polling is frequent. But if polling is infrequent, data dissemination is slow.

Salamander is a wide-area network data dissemination substrate designed to support push-based applications [MALA97]. 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.

Geographic routing sends messages to participating machines located in close physical proximity to destination points [NAVA97]. Primarily used to support mobile computing, geographic routing protocols may prove a useful component of a Revere system.

The Clearinghouse project at Xerox PARC [DEME87] 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, of which the most important and effective was "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 differences. First, Revere's scale is several orders of magnitude larger. Second, Revere must handle a more rapidly changing network topology than did the Clearinghouse. Frequent medium-term disconnections of "leaf" nodes are the norm, and Revere must deal with node and link failures on-line and automatically. Third, unlike the Clearinghouse, Revere cannot trust all its nodes completely. Fourth, Clearinghouse's operational constraints allowed for a daily six-hour window in which to propagate updates amongst the replicas. Revere must share all resources with normal demands, and it must propagate much more rapidly than daily in some pre-defined period. Fifth, Revere faces a much larger heterogeneity problem than the Clearinghouse did.

Several products and services such as Pointcast send individually customized information to large numbers of users periodically. Although Pointcast sends information to a large number of sites, it sends different information to different users using standard Internet protocols, and cannot take advantage of broadcast capability of networks.

Key distribution mechanisms also have some relationship to Revere's problem. 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.

1.4.2 Multicasting

Multicast services, such as MBONE [DEER89], [MACE97], [VENK97], 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 [FLOY95], and have no redundancy. A great deal of research has been performed on multicast, and many different multicast systems handle some of Revere's problems. However, no existing multicast service currently handles all of those problems, nor is any existing service obviously extensible in a relatively simple manner 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.

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 [CHAN84], [KAAS89], [MOSE89], [YAVA95], [LEVI96], [LIN96] seek to ensure that all receivers get complete and correct information, typically by acknowledgments for all packets, which leads to an ack implosion problem. The use of negative acknowledgments allows many multicasting systems to avoid ack implosion, but this technique is not suitable for Revere's needs, because a Revere node can not assert that it missed a disseminated security update or not. Repair-request methods are also used by some multicasting systems to solve this problem, but, like nacks, they require the receiving nodes to be aware that they missed something.

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 [GARC93]. The tree can be quite unbalanced in terms of performance for different senders. Recent work uses an overlay hypercube topology for multicasting, significantly improving the performance for any sender. [MOSE97] 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 [BIRM91] [WHET95], handling large numbers of acks [PING94], using minimal network resources [YAVA93], and managing the multicast groups [YAVA95], [LIEB97]. These objectives are different from rapid dissemination of a small amount of data to be sent to all nodes.

Generally speaking, services like IPSEC, reliable multicast, and broadcast may prove to be useful components of the Revere system. While they do not solve the entire problem, they do solve important subproblems in ways that allow for easy use or composition. We anticipate that many of the low-level needs of the Revere system will be met by use of these and other existing message delivery and network security services.

1.4.3 The Network Time Protocol

The Network Time Protocol solved a problem with some similarities to Revere's problem [MILL91]. 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 addressed 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 the NTP solved was different than Revere's in important ways. The NTP required manual configuration, while Revere must do automatic configuration. Moreover, a backup path can evolve into a primary path in NTP, a strategy which can work better in the NTP than 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 client node cannot locally distinguish between no Revere messages being sent and subversion preventing delivery of messages.

Also, NTP never required retransmission of missed messages. Time updates are, by their nature, ephemeral. A missed update was of no value shortly afterwards, so disconnected nodes had no need to obtain them.

Despite the differences, NTP may provide key architectural ideas for Revere in several areas.

1.4.4 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 release individually, resulting 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 Quarterdeck's TuneUp and Cybermedia's Oil Change. 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 one button push from the user. Both products are designed to ease 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 vise 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.

1.4.5 Replicated Data Management

Research in replicated data management, particularly optimistic peer replication, provides insight into methods of ensuring that different sites share the same view of the data. Client/server solutions [SATY90], [KIST91] and primary copy solutions [LISK90] 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 [GUY90], [PAGE98]. 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 [GOEL96]. Truffles, a related project, provided some forms of security for optimistic replicated file systems [REIH93]. Rumor provided a more portable version of the functionality of Ficus [REIH96]. Rumor 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 [RATN98], an issue of great relevance to Revere. 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 [WANG97]. This and other relevant replication research [BIRM85], [ALON89], [HISG90], [BADR92], [GOLD93] [DANZ94], [DEME94], [GRAY96], will help guide data consistency and dissemination issues in Revere, but this work is directed at a much more general problem than Reveres. Revere will be able to use simpler, lighter-weight solutions than file replication systems could.

1.4.6 Virus Signature Distribution

Virus signature dissemination has been one of the earliest practical purposes for security information dissemination. Many 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 and produces 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. Revere will be able to ensure wider, faster distribution of virus signatures and similar security information by using more of the available facilities of the network.

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 [KEPH97].

1.4.7 Intrusion Detection

Much research has been performed on intrusion detection [DENN86], [LUNT88], [SNAP91], [KIM94], [CROS95]. Methods that defend networks or cooperating distributed systems against intrusion, especially when all members are peers, are particularly relevant [MUKH94], [WHIT96], [ZERK96]. Our research is not competitive with these efforts. Rather, Revere will be a user of existing intrusion detection techniques. We expect that the practical experience of applying them to a new problem will uncover new possibilities and requirements which our work must handle.

1.5 Overview of the Prospectus

In Section 2 we will discuss the practicality by doing some analysis on attacks and also possible ways to fight against them. The attacks are categorized mainly according to the damaging result they caused. Section 3 will describe dissemination mechanisms we hope to build into Revere facility. There the high assurance by redundancy, scalability, receiver based policy, pulling, and working with existing transport mechanisms are discussed. Section 4 will talk about the purpose, automatic formation and adaptive management of RBone.

After these discussions, in section 5 we introduce a strategy doing dissemination and RBone management based on sending table concept. Then in section 6 the dissertation plan is outlined. Finally it's the summary of the prospectus.

2 Feasibility - Attack Analysis

2.1 Components of Revere System

There are mainly two components:

1. Dissemination source of security updates;

   The number of dissemination sources is small. The address of each source and other necessary information is already publicly known.

2. Revere node

   There can be two types of Revere nodes:

   - Middle Revere node, which receives security updates and also feed security updates to some other Revere nodes;
   - Leaf Revere node, which only receives security updates and does not forward the security updates.

   The number of Revere nodes, as we addressed before, is of huge amount.

2.2 Attack Model

The attacks could be by break-ins, worms, viruses, Trojan horses, vulnerabilities in system software, and so on.

However, to facilitate the design of our system, the best description of attacks must be the results caused by the attacks, not its form. It's the attacking result that a robust system design should essentially pay attention to. A specific attack may cause different kinds of damages, while one specific damage can be caused by different kinds of attacks.

According to the danger or damage caused by attacks, an attack could be (in the following a message could be a security update to disseminate or a message used for Revere control and management):

Corrupting a message

- Modification of message.
- Fabrication (forgery) of message.

Corrupting the transmission path

- Blockage of message dissemination
- Redirection of message transmission
- Denial of service(DOS) attack caused by message replay overloading

The blockage and redirection can happen when a machine on the transmission path to reach a Revere node is subverted. This machine can be a Revere node, or not (a router for example).

The DOS caused by replay is very special in Revere, particularly the replay of security updates, because Revere is essentially a mechanism for providing a tremendous fan-out of messages. One initiator of a security update will get his message delivered to thousands or millions of nodes. If not handled carefully, Revere could be misused to flood the network or Revere participant nodes. While Revere authentication mechanisms would ensure that no improper actions were taken on the basis of forged Revere messages used for denial of service attacks, unless some care is taken, Revere would tend to disseminate those false messages widely. Merely checking the authenticity of a message before sending it on would help somewhat, though it would certainly tend to slow down dissemination. However, attackers could replay legitimate Revere messages to cause these kinds of denial of service attacks. Thus, a Revere node should not just check authenticity of a message, but should also determine if this message has been replayed.

Leakage of secrecy

When a security update to disseminate needs to be treated as secrecy, one attack could be the leakage of the contents of the security update. Encryption is the best way to deal with this attack, but the key used to decrypt the security update then must be protected against those attacks corrupting the key.

The attacks could be internal or external. The internal attacks could be from subverted disseminating source, or a participant who wants to receive data. The external attacks could be all kinds of methods counteracting data dissemination, and it can happen not only toward a Revere component, but also toward a non-Revere component, for instance, a gateway on a path to reach some Revere nodes.

One assumption on the attacking model is, in some rare cases, a genuine machine may not get disseminated message because of serious attacks, for instance, when all neighbor Revere machines are corrupted. So, in terms of attack model, Revere is a best-effort dissemination system. Nevertheless, the isolated Revere machine may still detect some bizarre situation, for example, when it only can receive corrupted security updates no matter how many incoming paths it has. An intrusion detection mechanism might be built into Revere.

2.3 Fight Against Attacks

Corresponding to the attacks listed in section 2.2,

Corrupting a message through the modification and fabrication of security updates This can be prevented by signing the security updates. As long as the key management is successful, each Revere node can use the public key algorithm to verify the authenticity of a received security updates by checking the signatures.

Right now we assume the key management is a separate procedure, that is, it's out of the scope of Revere itself. We assume when a dissemination session is ready to run, the dissemination sources and all Revere nodes have got necessary keys.

However, as for the control messages used for Revere system management, it's not realistic to assume that the public key of a normal Revere node is already publicly known. They need to be handled differently, such as how to protect or be resilient against a modified or fabricated Revere control messages, or how to prevent an attacker from impersonating as another Revere node and initiating control messages, or at least limit the attack to be within some endurable scope.

Corrupting the transmission path by blockage, redirection, and DOS(Denial of service) by replays We propose to counteract this attack by two means. On the first hand, we want to build a dissemination mechanism on RBone through which each Revere node can specify its own policy fighting against the transmission path attacks, for example, it can specify the redundancy level of the in-bound paths. Through building redundancy into Revere, we hope there can be at least one path bringing authentic security updates timely to a Revere node, even when varying number of Revere nodes have been subverted. This will be particularly addressed in section 3.

On the second hand, we will investigate various methods of detecting replays and preventing their use for denial of service. One possibility is to maintain archives of Revere messages everywhere, and only forwarding messages not stored in those archives. These archives need not store the entire message. They only need to store its signature, since the purpose of this archive is merely to detect replays. Still, these archives need to do garbage collection. In order to avoid the replay of Revere messages not kept in the archive, attaching a reasonable time-to-live to a Revere message might help. Messages beyond their time-to-live could be rejected without consulting a replay archive, and replay archive entries could be flushed when the matching message's time-to-live expired. This solution would not work for the disconnected machine archive, perhaps, because some computers may be disconnected for lengthy periods.

In many cases, a disconnected node returning to the network will not connect directly to the repository it consults to pull missed Revere messages, but will go through several network hops to obtain those missed messages. Intermediate Revere nodes have probably already seen these messages, but they should not suppress them as replays. The Revere anti-replay facility would not be invoked, in this case, because Revere would not be involved in handling the messages at the intermediate nodes. Standard message delivery methods (e.g., IP) would use the nodes' underlying networking capabilities without ever involving Revere. This possibility does not open up new avenues for replay attacks, since the network already must permit normal IP routing.

The replay of Revere control message can also cause denial of service attack. For instance, when a Revere node keeps on sending another Revere node or even dissemination source some control messages. The prevention of this denial of service perhaps can not be prevented by archiving control messages. On the other hand, we hope there is no as much fan-out of control messages as that of security updates. This will depend on the detailed design of the management of RBone. We expect each control message can only be sent within a limited scope.

Leakage of secrecy This seems tough because of the scalability. If a different key must be used for sending the security updates to each Revere node from a dissemination source, then the dissemination source has to keep all the public keys of each Revere node, which is not scalable. If a shared key is used, then the 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 can not re-get the security updates disseminated after it left Revere.

On the other hand, the leakage of secrecy is not worried in some situations, for instance, when disseminating the signature of a newly discovered virus, where we'd like to see as more machines as possible to get the security updates of the new virus.

Our stance in handling the leakage of secrecy is to leave it as a secondary goal, while focusing on dealing with counteracting other attacks and doing a very satisfactory dissemination.

3 Dissemination Mechanism

3.1 HIGH ASSURANCE

Because of the importance of the security updates to disseminate and the possible attacks discussed in section 2.2, Revere nodes want high assurance that they can receive security updates if there is a disseminating session, even if the data has to come through a hostile environment with all kinds of possible attacks.

We claim that using acknowledgment or negative acknowledgment in Revere is not realistic. And we want to achieve the high assurance by redundancy.

USING ACKNOWLEDGMENT?

To achieve high assurance, one method is that each node acknowledges each received Revere message with a signed response. Who should receive the response and verify the authenticity of ack? Scalability consideration makes this highly difficult. Even with distributed or hierarchical method, in order that each verification server maintains all necessary keys, heavy overhead could be resulted, and the protection of verification servers brings a new problem.

Even if the ack verification is doable, what if the ack gets lost physically or dropped maliciously? An attacker who blocks the dissemination would also probably block the ack.

USING NEGATIVE ACKNOWLEDGMENT?

Many services that try to disseminate information to large numbers of nodes use negative acknowledgments to avoid the ack explosion. This approach only works when it is feasible for a receiver to be able to realize that he has not received a security update he should have received. 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, negative acknowledgment has same problem as ack in preventing itself from getting dropped.

USING REDUNDANCY?

If a node can receive data from multiple disjoint paths, only when all paths are corrupted will the receiver be prevented from getting the data.

While it seems this method wastes bandwidth, the disseminated data is usually of small size, low frequency but vital importance, and as long as the number of disjoint paths is not too high, using redundancy is acceptable.

The difficulty is how to compose or detect disjoint paths. If there is no absolute disjoint paths, is it possible to find approximately disjoint paths? How to deal with choke point?

3.2 SCALABILITY

One obvious characteristics of the dissemination of security updates is its large scale. As we said in section 1.2, the scalability of Revere must be considered from day one. It must match the scale of Internet very well if Revere aims to run on top of Internet.

3.3 RECEIVER BASED POLICY

Each Revere node, as a receiver of security update, is probably heterogeneous in terms of communication characteristics and platform. Moreover, each receiver may be running a different application which may have different request of using Revere service. Thus, it's possible that each receiver may have different degree of requirement on the assurance and other aspects of the dissemination. Even with respect to one particular receiver, it may have different requirement on different data. One immediate implication of this is that each receiver may like to have its own level of redundancy of hearing security updates.

3.4 PULLING

When security updates is pushed from a dissemination source to large number of connected machines, some machines may not be connected or temporarily turn off its capability of receiving Revere data. When they regain the connectivity, they still want to get the missed security updates.

Some reliable transmission mechanisms, such as TCP or reliable multicast, can endure a short period disconnection. But a longer disconnection period will result in a timeout of the transmission, which makes the Revere node miss the security updates.

A more general method of handling the problem is to have disconnected nodes inquire about security updates that occurred during disconnections.

But from where? The source can not re-disseminate the missed data to each machine again, because the missed data by each machine could be different, and the large scale of the system makes this problem harder. So some designated nodes would have to maintain a repository of old Revere messages that could be used to respond to inquiries. This requirement immediately leads to the issue of how can such a repository be pruned, so that very old Revere messages do not clutter up the storage space of repository machines. Answers to this garbage collection problem themselves have severe security requirements, as an intruder who could improperly force garbage collection of a vital message could effectively prevent delivery of that message to disconnected nodes that rely on the archive. We plan to use long timeouts on security updates, backed up with repositories that store all updates forever, to deal with this problem.

The repository node receiving the inquiry might very well be subverted itself, leading to further problems. The methods already discussed for authenticating Revere updates can be used to ensure that a subverted node responding to an inquiry cannot forge false Revere messages, but the subverted repository could easily fail to deliver some of the updates it had received. This intentional failure would effectively deny the reconnected portables the protection implicit in receipt of those messages. Revere must allow portable computers to obtain some degree of certitude that they have received all messages missed during disconnections.

To fight against this attack, we again plan to use the redundancy to achieve the high assurance. One possible approach is to have each reconnected node receive updates from one trusted repository, then verify that they have obtained the complete set by consulting one or more other repositories.

3.5 WORK WITH EXISTING TRANSPORT MECHANISMS

The dissemination system should work with existing networks, requiring no changes to those networks. Adding new features to network routers or gateways in order to support Revere functionality is unrealistic.

On the other hand, the dissemination will certainly make use of existing protocols and communication capabilities. Opportunistic use of available network resources or capabilities, such as wireless communication, broadcast by a satellite, multicast over a LAN or WAN, should be considered.

The opportunistic use has two aspects:

- *Rapid dissemination* When there are several different alternatives to reach a Revere node, one of them may be preferred for the seek of speed.
- *Resource usage* However, the resources must be used in a reasonable way.

Heterogeneous participant nodes, in hardware, software, operating systems, and even communications capabilities, increase the difficulty of the problem. Also, the necessity to handle different types of security information and support different levels of trust among system components complicates the solution. In a word, we have to make trade-off between the speed and the resource usage, while taking into consideration of each Revere node's particularity.

3.6 GETTING CONFIDENCE

Determining in real time that how widely the data has been disseminated is a hard problem in a large scale environment. A dissemination source may even not know the number of currently connected Revere nodes. Particularly not all messages can be trusted. To have an on-line feedback mechanism needs some clever design. This is a necessary mechanism that has to be implemented to make Revere work, although it may be required in some situations. We will consider the possibility of solving this problem, but of second priority.

4 RBone

The dissemination mechanism discussed in section 3 must be based on a RBone which meet the features of the dissemination mechanism.

We assume that RBone is composed of Revere nodes and the logical links among them. A dissemination source is not necessarily an element of the RBone, though it knows how to disseminate the security update through RBone based on its partial knowledge of RBone.

There are three aspects of handling the RBone:

- how to utilize the RBone;
- formation of RBone; and
- management of RBone;

4.1 Using RBone

As said in section 2.1, a Revere node can be either a middle Revere node or just a leaf Revere node. One possibility is that there is no middle Revere node in Revere, that is, each Revere node is not responsible of forwarding a Revere message. Is this possible or feasible?

The answer is no. For reasons said in section 1.4.2, we will not use multicast as a universal service for our dissemination service, although we may use it in an ad-hoc fashion. So, if there is no middle Revere node, then each dissemination source has to store the address of all Revere node and transmit security updates to Revere nodes one by one. This is obviously not scalable, not efficient, and hard to provide redundancy in transmission.

So, the conclusion is that RBone will be composed of Revere nodes, with middle Revere nodes "inside", which can forward security updates to other Revere nodes, and leaf Revere node "outside".

4.2 Formation of RBone

The formation of RBone starts when the very first machine wants to be a Revere node. Assuming the address and other necessary information of dissemination sources are already publicly known, then the question is how the newborn can contact the dissemination sources to add himself into the RBone to become the first Revere node.

The formation of RBone continues whenever there is a new machine to join or an existing Revere node to leave.

The key to the formation procedure is that each new joined Revere node can get more than one incoming paths (if available) to receive security updates.

Also each machine may have its own request for the properties he'd like to have by utilizing the functionality provided by Revere. Typically, a machine can specify its acceptable resiliency level.

Moreover, each machine is heterogeneous in that it has different communication capability such as having high or low bandwidth, being able to listen to or send multicast or broadcast messages, etc.

So, the formation procedure should take into account of these factors so that the later dissemination can utilize them.

4.3 Adaptive Management of RBone

The management of RBone must be adaptive automatically to the changes of RBone, for example, when a new Revere node joins, or when an existing Revere node leaves, or when the characteristics of the connection between two Revere nodes changes, and so on.

The management of RBone also has to be distributed. Each Revere node only has partial information of the whole RBone, probably most of its neighbor Revere nodes. It has to be able to detect some changes based on some phenomena.

Being able to detect changes relevant to RBone management has to be detected rather quickly. We may use the heartbeat mechanism to keep each Revere node to be aware of its neighbors and the characteristics of the link between itself and each neighbor. However, it's also need consideration how to detect new feature of the RBone because of some radical changes, for instance, after a new Revere node joins Revere, how those existing Revere node nearby can begin to contact with the new Revere node?

To evolve the RBone based on detected changes is challenging. Because of the scale of RBone, each Revere node has to be able to respond to changes in an asynchronous fashion.

5 One Receiver-based Multi-path Strategy

In this section we propose one strategy doing dissemination. It does not mean we'll follow exactly the procedure here to do our later work on dissemination, but rather to convey the efforts we have made and the understanding we have got on solving the dissemination problem. By addressing and analyzing this strategy, we hope we can find more issues to study and provide more insights on the design of a brilliant dissemination solution.

5.1 Dissemination Based On Sending Table

The dissemination based on sending table works as following: Each Revere node, including the source, keeps a local sending table, which specifies a list of hosts and their related properties. When a source wants to disseminate some security update, it finds a list of hosts from its local sending table, and sends the security update to all of them. The way of sending the security update to each host can be influenced by the properties described in the sending table for the host in question. For example, the sending can be by broadcast or multicast to a list of hosts, or the security update is sent using IP source routing to a host (even through more than more physical routes specified by the source routing).

Similarly, when a Revere node receives a security update and its own local sending table is not empty, it will use the same procedure as used by the dissemination source to feed the security update to those hosts listed in its sending table.

5.2 Building Sending Table

Then the question is how to build the sending table on the dissemination sources and each Revere node (the leaf Revere node will have an empty sending table).

We can see from the procedure above that the sending tables associated with the dissemination sources or

Revere nodes link the components of Revere together, and the contents of sending tables lead to an unique composition and structure of RBone.

So, building sending table is exactly the formation of the RBone.

Building the sending table should follow the principles of scalability, adaptability, heterogeneity, and with high assurance. Finally we won't forget security.

5.2.1 Procedure

The procedure we propose to build sending tables is receiver-driven:

1. *newborn:* When a newborn wants to join Revere, it sends a "Join Request"(JR) toward the source it's interested. We assume the necessary information of a small number of dissemination sources are publicly known. Some info included in a JR could be newborn's request of resiliency, its communication properties, etc.

   For instance, the newborn may request to build three incoming paths for him if possible. And it may also declare that it can hear multicast message at some multicast address, and it can hear broadcast message to its subnet. Or it may claim it's connected to Internet by modem and another machine is his proxy. These information is useful in building an entry for the newborn in some sending tables where these information can be stored there for the seek of later dissemination.

2. *source:* The source decides whether to simply add an entry into its local sending table corresponding to the newborn, including some properties from the JR.

   Normally, for scalability reasons, the dissemination source has to make sure that its sending table is of limited size, such that usually it can not keep an entry for each Revere node in the sending table. Moreover, if a newborn asks multiple paths to be established, simply adding one entry is not enough since this only means a single path directly from the dissemination source to the newborn.

   So the entry will be added only when the sending table of the dissemination source is of small size and the newborn also just requires one single path to hear security updates. In most cases, the source will just go to the next step instead.

3. *source:* The source will run a *recommending algorithm* to recommend some Revere nodes to let the newborn to consult with. Each recommended Revere node is a candidate through which the newborn will hear security updates.

4. *newborn:* To decide which several Revere nodes from the recommended ones that the newborn will hear security updates, the newborn tries to detect related information from each recommended Revere node and the source, and tells the source these information.

   The related information can be the distance between itself and the recommended Revere nodes, and the metrics can be hop count or round trip time. It can also be security level or geographical position. It can also be combination of them, etc.

5. *source:* To decide which Revere nodes to select to forward Revere messages for the newborn, the source runs a *decision making algorithm*, and it will notify the newborn which recommended Revere nodes are finally selected.

   If the source itself is also selected, an entry for the newborn will be established with related properties. In this case, table replacement may happen. A *sending table replacement algorithm* is needed to run since the effects of replacing an entry might propagate to many other Revere nodes if not handled properly. The simplest is to let the newborn act as the feeder of the replaced.

6. *newborn:* The newborn hears the notification message about which Revere nodes are selected.

   If the notification says that the newborn needs to contact some selected Revere nodes, each of which can forward security update to the new born, the newborn sends an "Enroll Request"(ER) to each of them.

   If the notification says that an entry has been added to the sending table for the newborn and no necessity to contact more Revere nodes, then this notification message serves as an acknowledgment for one established path, and no further ado needed for handling it.

   The newborn can assert himself that he already joined Revere when all messages sent to the dissemination sources or Revere nodes have been responded, and all notification messages are of the second type.

7. *Revere node:* Each Revere node will respond to ER by recursively repeating the procedure that the source did for the JR, i.e. all the above steps.

8. *newborn:* The newborn will retry the above steps if it times out for hearing the messages from the dissemination source about the recommended Revere nodes, or the messages about the selected Revere nodes, or notification.

   If it times out for hearing the same messages from Revere node, the newborn may ask supplementary work to be done, although it certainly can retry from the very beginning. For example, if no response for a ER, it might just try to send an ER to an once *recommended* Revere node.

5.2.2 Flexibility

One important flexibility of the steps building sending tables is that, if a newborn knows there is a Revere node nearby, it can choose to send "Enroll Request" to that Revere node directly, without bothering the source. This can greatly liberate the source from answering the "Join Request"s. (More generally, a newborn can send request to any Revere node to join Revere, as long as he knows the address of that node and it likes to do so.)

In a word, a newborn has options to join Revere "from top down", "from bottom up", or "from middle". We need to consider more to choose one of them, or allow all of them but use each one depending on real situations.

5.2.3 Three algorithms

In the above, three algorithms are used, i.e. recommending algorithm, decision making algorithm, and sending table replacement algorithm. Right now we don't know exactly how they should work, however, there are indeed some properties that we'd like to achieve for each algorithm, and some questions we must answer.

Recommending algorithm This algorithm decides what Revere nodes *may* work well if used as one hop for forwarding a security update along a potential path to the newborn, based on the characteristics of the newborn included in its JR.

The algorithm must take into consideration of the newborn's request for the quality of its incoming paths.

There can be different criteria. Do we like to choose those in the nearest neighborhood of the dissemination source, or those in the nearest neighborhood of the newborn, or those machines with special functionalities such as multicasting or broadcasting using satellite? Does it make sense to consider the geographical position of a Revere node? How can we utilize the IP address of every involved machines?

There can also be some properties that we don't like the recommended nodes to possess. For instance, a Revere node with very thin bandwidth, or significant mobility, or low degree of connection, or weak trustfulness.

Decision making algorithm This algorithm selects which Revere nodes *can* work well if served as one hop on approaching the newborn when disseminating the security updates.

This time the algorithm will have to make decision based on the data collected by the newborn about all the recommended Revere nodes *and* the JR of the newborn, in order to choose some of them based on some criteria.

The criteria can be similar to those used in the recommending algorithm.

Sending table replacement algorithm This algorithm runs when an entry in a sending table has to be replaced, and it decides which entry should be replaced. First in first out? Or last in first out? Or based on some criteria?

After an entry is replaced, the host corresponding to that entry will not be able to get security updates from the sending table owner any more. But since the host may also take responsibility of forwarding security updates, this means some hosts may lose one path to receive security updates. This corresponds to the link failure in the RBone. How can we compensate the loss? Can it work well always to add one link between the host corresponding to the new entry and the host corresponding to the replaced entry?

5.3 Maintaining Sending Table

After a sending table is built, it needs maintenance. Each Revere node is expected to hear heartbeat from its feeders, from which the Revere node directly get Revere messages. If not, it'll time out to try to find new feeders. So, if a Revere node quits, other Revere nodes depending on this Revere node directly or indirectly may find a different path.

A feeder will not send heartbeat if it times out, that is, if it has not hear heartbeat from its own feeders for a rather long time. Otherwise, it's misleading because other Revere nodes may still believe their feeder is capable of receiving security updates and feeding to them.

For a healthy Internet body, the heartbeat rate can be slow. But if the heartbeat rate is zero, there is no adaptability, as there is no liveness if there is no heartbeat in a human body.

A difficult problem is how to improve the quality of paths for a Revere node by improving the sending table. Each Revere node likes to have a shorter path, or even be able to get messages from the source by just one hop for instance, however, each Revere node, including the source, can not have a sending table without limits. This implies a Rever node may have to use another Revere node as its feeder which may result in a slow path. Moreover, what if a new Revere node joins later, and this new node may feed some Revere nodes with better quality?

The simplest way is to let each Revere node to periodically try to re-join Revere, so that in the next join a better path can be detected. The drawback of this method is that its overhead can be a lot. But if we can let each Revere node re-join *from bottom up* and still be able to detect better path, we can achieve good scalability and low overhead.

5.4 Imposing Security

All above procedures can be complicated by the security requirement. Seems no matter what dissemination algorithm to use, there is a security problem of handling fake or malicious control messages.

If a Revere node is already subverted, it may persuade a newborn to utilize him as one hop on one or more paths to reach the newborn, probably by showing some attractive but fake characteristics. Since it's hard to tell whether a recommended or selected Revere node is genuine or not, by assuming that large percentage of Revere nodes are genuine, we might be able to avoid this through redundancy.

Another attack in the sending table based strategy is the denial of service attack by keeping on sending some control messages, such as JRs, ERs, or other messages.

One more bad thing could be that an attacker can impersonate as another Revere node to send all kinds of Revere control messages described in section 5.2, to mess everything up.

6 Dissertation Plans

The following describes the tasks we expect to fulfill, based on our current understanding. Obviously in following research, we may find some missteps, or unanticipated chances to better our approaches, or we may have to make some tradeoffs. However, the following description will serve as a blueprint, ensure we have considered all aspects of the project, and give readers a understanding the amount of effort involved.

6.1 Basic Revere Functionalities

The first step is to build basic Revere functionalities which can show how Revere works. This is also the most important part. Particularly, the following will be designed with necessary implementation.

6.1.1 Dissemination mechanisms over RBone

As reported in section 3, for disseminating security updates, we want to achieve high assurance through high redundancy, scalability, opportunistic use of heterogeneous transport mechanisms, security updates pulling by disconnected Revere nodes, and so on. As we reported in section 4, this will need scalable and adaptive method to form and manage RBone. We divide the work into the following several tasks.

Security updates delivery analysis We have gained pretty much understanding in terms of the fashion and characteristics of delivering security updates. The further work to do is to understand more typical scenarios and get more insights.

(Major) Task 1 *Analyze typical or acceptable behavior and characteristics of dissemination source, and how they make their necessary information available, such as public keys. Maybe some works don't need to be done by Revere, but we should have clear expectations or assumptions of our dissemination sources to facilitate the design of Revere itself. We also need to analyze Revere related behavior of a Revere node for the same purpose.*

RBone formation and management We addressed our current sending table based strategy of doing this in section 5. As we reported there, this strategy is still being studied. This task is the kernel among the basic Revere functionalities.

The formation of RBone has to meet the dissemination mechanisms described in section 3, particularly, each Revere node should be able to have multiple paths, if available, to receive security updates. The scalability must be considered as an important goal too.

Two aspects must be considered:

- *Active Revere node* Each Revere node is *active* in that it can specify its own requests when joining

Revere, for instance, it can specify what level resiliency it likes to have in terms of the number of the in-bound paths. It can also describe its own Revere related information to let Revere make better use of its capability in doing dissemination or security enforcement. Moreover, a Revere node can actively coordinate with other Revere nodes, particularly those in its neighborhood, to forward security updates. A Revere node can also monitor some changes in the Revere system to tune the system automatically, for instance, when a path is broken or had better or worse performance than before, or when a new path is added to the RBone. In a word, each Revere node is not limited to be just able to hear security updates passively ever since it joins Revere.

- *Opportunistic use of heterogeneous data transport mechanisms* Revere must take into consideration that Internet is composed of heterogeneous components, especially the transport mechanism. To use a unified transport mechanism may be able to simplify the design and implementation, but this may waste some resources but still only bring poor performance. Opportunistic use of resources can also provide some features based on some special transport mechanisms.

  Revere need to understand fundamental differences in types of data transports and adapts Revere's behavior to match those differences. For example, floppy disk and other hard media transport mechanisms tend to be one-way, so Revere would assume that no back-flow over that channel is possible. Revere would need an understanding of the delay characteristics of various data transports, to allow estimation of the rapidity of the spread of a Revere message. (In some cases, this information may be available through measurement, rather than understanding the specifics of the hardware and software implementing the link. Some transports, like floppy disks, however, are difficult to measure automatically.) At its most basic, this mechanism would determine whether a message was sent out with a suitable IP header, or as an Ethernet broadcast message, or as a file stored on a floppy disk, or in some other format. This component will consist of the necessary system support for handling all forms of data transport to be used by Revere. This component also will contain software capable of examining the current status of various local transport mechanisms available at a node.

Certainly the security of RBone formation and management must be considered simultaneously while the procedure is designed and implemented. We purposely leave the RBone security enforcement out of this task and assign it to a separate task, where it's addressed with other security issues. This does not mean we'll design security separately, this only mean that security is important.

(Major) Task 2 *RBone formation and management. The dissemination of security updates must be able to provide high assurance through the redundancy by having multiple in-bound paths for Revere nodes. It should be scalable, receiver-driven, and be able to incorporate heterogeneous transport mechanisms.*

Pulling by disconnected Revere node Although without being able to pull missed security updates by a Revere node, we still can show a Revere dissemination session, but a Revere system without pulling capability is incomplete in that a disconnected Revere node has no assurance of receiving complete security updates. So, our next major task in Revere dissemination mechanisms is to design and implement the pulling functionality.

(Major) Task 3 *Pulling missed security updates. This includes configuration of security updates archive, publicization of the archive site, garbage collection of the archive, protection of the archive, the procedure of pulling the missed security updates from archive, and the procedure of contacting multiple archives to achieve high assurance.*

6.1.2 Security enforcement

As reported in section 2, Revere must fight against attacks corrupting message, corrupting transmission path, or leaking secrecy. While using redundancy built in RBone to achieve some level of security, sophisticated security enforcement is still needed. As we discussed in section 2.3, the security enforcement can be divided into several major tasks. The intrusion detection of Revere itself is listed as an optional task.

(Major) Task 4 *Design methods to fight against the attack corrupting the security updates or Revere control messages. Security update can be protected from modification and fabrication through signing by the dissemination source. The protection of Revere against the corruption of Revere control messages is also to be handled, but probably in a different way.*

(Major) Task 5 *Design methods to protect against transmission corruption by blockage, redirection, and denial of service by replays. Whereas the built-in redundancy in RBone can serve for this purpose, the replay needs to be prevented. The replay of security updates, as reported in section 2.3 can be prevented by maintaining an archive of security updates, and each security update has a time stamp. The replay of Revere control messages needs to be handled differently.*

(Optional) Task 6 *If secrecy of the security update to disseminate needs protection, a method of counteracting the leakage of secrecy is needed. Since it's unrealistic to separately encrypt the security update with each Revere node's public key, we may need to consider a method based on session key shared among all Revere nodes.*

(Optional) Task 7 *Intrusion detection of Revere itself. It's about how to detect attacks toward Revere through some abnormal behavior.*

should I include this?

6.1.3 On-line feedback

It is an optional task to determine on-line and relatively quickly the degree to which a given Revere update has been disseminated to the target nodes. Two factors contribute to the difficulty of this problem:

Scalability To collect feedback from a large amount of Revere nodes quickly must address this issue.

Hostility Environment Not every feedback can be trusted. It can be from a subverted Revere node which tries to persuade the system to believe that the dissemination is successful.

(Optional) Task 8 *Determine on-line and relatively quickly the degree to which a given Revere update has been disseminated.*

6.2 Test the system

After the Revere prototype is implemented, we need to test the effectiveness of Revere. This will be done first by designing a testbed. The testing of Revere will have to be small scale unfortunately. The large scale of Revere will have to be analyzed by doing simulation.

The testbed will be composed of heterogeneous machines, that is, each machine may have different communication capabilities. They may also be over different platform. The connection characteristics between any two machines can also be arbitrary.

The testbed will be able to cause some security attacks as reported in section 2.2.

(Major) Task 9 *Construction of Revere Testbed and Evaluation of Revere. This will include survey of available machines and construction of a testbed based on test purpose of Revere. Then we will artificially cause some attacks to see how Revere responds. The result will be used to improve our Revere prototype.*

6.3 Simulation

Simulation allows testing of Revere at scales several orders of magnitude greater than that possible in a testbed. A study of scaling and other properties of Revere derived from use of the simulation can perhaps help us to locate some problems we can't foresee before using it, or it can ensure whether our design and implementation of Revere mechanisms is correct.

(Major) Task 10 *The simulation of Revere. This includes the investigation of possible simulation platforms, design the simulation (especially for the large scale), implement the simulation by modeling every important Revere components, run the simulation and collect data, and do analysis.*

6.4 A Usable Product

Cryptographic and authentication interfaces When a client of Revere wants to get disseminated security updates based on the services that Revere provides, it may have its own special security requirements. It would be good if Revere can provide a well specified and flexible cryptographic and authentication interfaces. This will need a wide investigation of the requirements of most potential Revere clients.

(Major) Task 11 *Design Revere cryptographic and authentication interfaces so that a Revere client can get Revere service while still satisfying its own security requirements.*

Revere policy mechanisms This will consist of interfaces that deal with the local actions a node takes in response to the arrival of a security update. Different security systems using Revere will require different actions to be taken when a message arrives.

In the case of a security update used in a distributed intrusion detection system, the local intrusion detection software would like to be notified of the update on the interface it provides for that purpose. In the case of a message to be logged, it would be appended to the particular log file. In the case of a new virus signature, the signature would be placed into the virus detection software's repository of signatures, and very likely the virus detection software would initiate a scan of local files for the new signature. In the case of a newly detected offending or kid-prohibited web site addresses, the address would be added to the list of those addresses of the software protecting kids. The interface will probably consist of general call-back facilities that allow client security systems to register a program or routine they wish to be executed when a particular kind of message arrives. This component will also subsume issues of the methods for determining which client system should handle particular Revere messages.

(Major) Task 12 *Design of Revere policy interfaces. This may include investigation of Revere clients of their best way to take advantage of Revere service, some sample policy mechanisms exercising those interfaces, as well as design and implementation of the methods used to determine Revere message type.*

Integrated feasibility demonstrations including Revere This is listed as an optional task. It's concerned about how to design a demonstration in which Revere is used as a supporting service, not a centerpieces, for other applications. The integrated work will be used to show how Revere can enhance the capability of some important applications.

(Optional) Task 13 *Integrated demonstrations with Revere as a supporting service. This includes identification of a potential Revere client, design the demonstration in which the client is supported by Revere, and run the demo to see if the client gets significant support from Revere.*

7 Conclusion

We have presented a design for large scale dissemination of security updates over Internet. Besides the scalability, we addressed potential attacks of the system and our proposal to fight against those attacks with some issues still unanswered. We also discussed the dissemination mechanism, including the high assurance through redundancy, receiver based policy, opportunistic use of heterogeneous transport mechanisms, and so on. The formation and management of RBone is also discussed, with one strategy explained.

We believe this system will be successful and provides a useful facility for Internet applications. The success of this system can save many applications to design their own dissemination solution by having our system as a basis.

We believe this system will also provide some insight in designing a dissemination solution in support of those applications which has no or little concern for the security.

There are some works that we have to leave for future researchers to investigate. We have include many future works in section 6 and list them as optional tasks.

References

[ALON89] R. Alonso, D. Barber, and S. Abad, "A File Storage Implementation for Very Large Distributed Systems," *IEEE Workshop on Workstation Operating Systems*, September 1989.

[BADR92] B. Badrinath and T. Imielinski, "Replication and Mobility," *Proceedings of the 2nd IEEE Workshop on Mobility of Replicated Data*, November 1992.

[BANN97] 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.

[BAUE92] M. A. Bauer and T. Wang, "Strategies for Distributed Search", *Proceedings of the 1992 ACM Computer Science Conference on Communications*, 1992.

[BERT92] D. Bertsekas and R. Gallagher, *Data Networks*, Prentice Hall, 1992.

[BIRM85] K. Birman, "Replication and Availability in the ISIS System," *Proceedings of the ACM Symposium on Operating System Principles*, December 1985.

[BIRM91] K. Birman, A. Schiper, and P. Stephenson, "Lightweight Causal and Atomic Group Multicast," *ACM Transactions on Computer Systems*, Vol. 9, No. 3, 1991.

[CHAN84] J. M. Chang and N. F. Maxemchuck, "Reliable Broadcast Protocols," *ACM Transactions on Computing Systems*, August 1984.

[CROS95] M. Crosbie and G. Spafford, "Defending a Computer System Using Autonomous Agents," *Proceedings of the 18th National Information Systems Security Conference*, 1995.

[DANZ94] 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.

[DEER89] S. Deering, "Host Extension for IP Multicasting," RFC-1112, August 1989.

[DEME87] A. Demers et al, "Epidemic Algorithms for Replicated Database Management," *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, 1987.

[DEME94] 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.

[DENN86] D. Denning, "An Intrusion Detection Model," *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, 1986.

[FLOY95] 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.

[GARC93] J. J. Garcia-Lunes-Aceves, "Loop-Free Routing Using Diffusing Computations," *IEEE Transactions on Networking*, V. 1, n. 1, February, 1993.

[GOEL96] A. Goel, "View Consistency for Optimistic Replication," UCLA Technical Report CSD-960011, Feb. 1996.

[GOLD92] 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.

[GOLD93] 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.

[GRAY96] 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.

[GUY90] 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.

[HISG90] 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.

[KAAS89] M. Frans Kaashoek, A. Tannenbaum, et al, "An Efficient Reliable Broadcast Protocol," *ACM Operating Systems Review*, Vol. 23, No. 4, October 1989.

[KEPH97] J. O. Kephart, G. B. Sorkin, D. M. Chess, and S. R. White, "Fighting Computer Viruses," Scientific American, Nov. 1997.

[KIM94] 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.

[KIST91] Kistler, J. and Satyanarayanan, M., "Transparent Disconnected Operation for Fault-Tolerance," *ACM Operating Systems Review*, Vol. 25, No. 1, January 1991.

[LEVI96] B. Levine, D. Lavo, and J. J. Garcia-Luna-Aceves, "The Case for Reliable Concurrent Multicasting Using Shared Ack Trees," *Proceedings of the ACM Multimedia Conference*, November 1996.

[LIEB97] J. Liebeherr, B. Sethi, "A Scalable Control Topology for Multicast Communications," Personal Communications, Polytechnic Univ. of New York, Brooklyn, 1997.

[LIN96] J . C. Lin, S. Paul, "RMTP: A Reliable Multicast Transport Protocol", *Proceedings of IEEE INFOCOM*, pp. 1414-1425, March 1996.

[LISK90] 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.

[LUNT88] T. Lunt, "Automated Audit Trail Analysis and Intrusion Detection: A Survey," *Proceedings of the 11th National Computer Security Conference*, October 1988.

[MACE97] M. Macedonia and D. Brutzman, "MBONE: The Multicast BackBONE," http://www.mice.cs.ucl.ac.uk/-mice/mbone_review.html, 1997.

[MALA97] 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.

[MILL91] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Transactions on Communications*, Vol. 39, No. 10, Oct. 1991.

[MOSE97] 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.

[MOSE89] Y. Moses and G. Roth, "On Reliable Message Diffusion," Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, 1989.

[MUKH94] B. Mukherjee, L. Heberlein, and K. Levitt, "Network Intrusion Detection," *IEEE Network*, May/June 1994.

[NAVA97] J. Navas and T. Imielinski, "Geographic Addressing and Routing," *Proceedings of the Third ACM/IEEE International Conference on Mobile Computing*, 1997.

[PAGE98] 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," to appear in *Software Practice and Experience*, 1998.

[PING94] 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*, v22 n1, p221-330.

[PRES97] The President's Commission on Critical Infrastructure Protection, *Critical Foundations: Protecting America's Infrastructure*, October 1997.

[RATN98] D. Ratner, "Roam: A Scalable Replication System For Mobile and Distributed Computing," UCLA CSD Ph.D. dissertation, January 1998.

[REIH93] 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.

[REIH96] 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.

[SATY90] 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.

[SNAP91] 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.

[VENK97] 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.

[WANG97] A. Wang, P. Reiher, and R. Bagrodia, "A Simulation Framework for Evaluating Replicated Filing Environments," submitted to *ACM Transactions on Computers*, 1997.

[WHET95] B. Whetten, S. Kaplan, and T. Montgomery, "A High Performance Totally Ordered Multicast Protocol," *Proceedings of IEEE INFOCOM*, 1995.

[WHIT96] G. White, E. Fisch, and U. Pooch, "Cooperating Security Managers: A Peer-Based Intrusion Detection System," *IEEE Network*, Jan/Feb 1996.

[YAVA93] R. Yavatkar and L. Manoj, "Optimistic Strategies for Large-Scale Dissemination of Multimedia Information," *ACM Multimedia 93 Proceedings*, 1993.

[YAVA95] R. Yavatkar, J. Griffioen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," *ACM Multimedia 95 Proceedings*, 1995.

[ZERK96] D. Zerkle and K. Levitt, "NetKuang - A Multi-host Configuration Vulnerability Checker," *Proceedings of the 6th USENIX Security Symposium*, July 1996.