Chapter 10
Anti-fragility to Malware Spreading

To achieve anti-fragility to malware spreading, this chapter applies the fail fast principle from Chap. 4 to the robust malware-halting technique developed in the two previous chapters. According to the fail fast principle, it is necessary to learn from failures in complex adaptive systems when the impact of the failures are still small. In the case of infectious malware epidemics, once malware is detected on a node in a networked system, other nodes infected by the same malware should be healed and susceptible nodes should be protected from future infections of this malware.

The two previous chapters showed how software diversity and hub immunization could halt malicious software or malware from spreading. This chapter combines compiler-generated software diversity [24, 56, 89], hub immunization, and imperfect malware detection/removal to achieve anti-fragility to the spreading of various types of malware in networked computing systems. The cloud is used to efficiently combine these techniques. The suggested malware-halting technique scales to huge networks because it does not require any tightly coupled interactions or adaptations between groups of devices. The average fraction of infected nodes is reduced compared to the examples in Chap. 8. The technique is of practical interest because malware is an omnipresent and serious security threat [21, 22]. The ideas in this chapter were first presented in [90].

We again study infectious malware, that is, computer worms with different spreading mechanisms. E-mail malware spreading via address lists and mobile phone malware propagating over short-range wireless links generate patterns of infected devices defined by sparse graphs [85], while malware scanning Internet protocol version 4 (IPv4) addresses at random produce dense graphs. We concentrate on sparse spreading networks in this chapter. Rather than trying to accurately model the spreading of real malware instances, we again analyze worst-case spreading where the first attempt to infect a susceptible device always succeeds.

Non-infectious malware strains, such as trojans, spyware, adware, and ransomware, mistakenly downloaded by computer users are viewed as infectious malware with limited spreading ability. We measure malware spreading in a networked computing system by the fraction of infected devices. A system is *fragile* to malware when small outbreaks of different malware strains spread to a large fraction of the devices. If the malware strains only spread to a small fraction of the devices, then the system is *robust*. A system under repeated attacks from malware is *anti-fragile* if it first learns to reduce the fraction of infected devices and then manages to keep the fraction small when the malware's spreading mechanism changes.

To achieve anti-fragility in practice, it is advantageous to build on existing and planned automated software mechanisms. We combine compiler-generated diversity, software downloads from application stores, hub immunization, and imperfect malware detection/removal to achieve anti-fragility to malware spreading. An agent-based model randomly adds software diversity to a software monoculture to create a software polyculture with a much reduced fraction of infected devices. The model demonstrates that periodically removing executable code, including unknown malware, from devices and installing new diverse code drastically increase robustness to malware spreading. If imperfect malware detection is added, the model gains a degree of anti-fragility because it can more quickly remove malware and update vulnerable code to keep the fraction of infected devices very small, even when the malware strains have unknown and time-varying spreading.

10.1 System Model

As in the two previous chapters, we study a network of interconnected computing devices and consider the devices at the operating system (OS) and application levels. Application stores in the cloud, such as Google Play and iOS App Store, utilize compilers with "diversity engines" to generate binary images for a huge number of devices, producing many different executable images from a much smaller set of OSs and application source codes [24, 89]. As in earlier chapters, we assume that a program's many binary images can be divided into classes such that all members of the same class have a common exploitable vulnerability, while members of different classes have no common exploitable vulnerabilities.

The number of classes measures the program's diversity, assuming roughly equally large classes. Since compiler-driven diversity promises to provide large diversity [24, 77], we forgo any notion of central control over the assignment of software diversity to computing platforms and make no attempt to minimize the use of diversity. This allows us to study the benefit of software diversity in systems with millions of devices. Cloud-based compilations of source codes allow application stores to support large numbers of download requests each day.

A significant fraction of all malware infections is not discovered by traditional signature-based malware detection because modern malware utilizes time-varying code obfuscation to avoid detection based on fixed byte patterns [21]. Emerging cloud-based anti-malware solutions promise to improve automated malware detection [22, 91, 92]. Servers in the cloud deploy heuristic, behavioral, and signature-based techniques to detect different types of malware by processing data collected by clients running on user devices. A cloud solution can also incorporate knowledge from other sources, such as malware honeypots, that is, computers capturing malware.

Despite the protection promised by cloud anti-malware, it is nearly impossible to keep all computing devices in a networked system free from malware at all times. The difficulty of detecting encrypted malicious traffic and the successful use of rootkits to hide malware suggest that automated malware detection will remain imperfect for the foreseeable future [21]. A more realistic goal is to provide a form of "community immunity," where most devices are protected against malware because there is little opportunity for outbreaks to spread. Whereas community immunity usually entails the immunization of nearly all entities in a population, we mainly deploy compiler-generated software diversity to reduce malware spreading. Our goal is not to force the fraction of infected devices to zero but, rather, to keep it very low over time.

We study consecutive outbreaks of different malware types, called *multimalware* outbreaks, because the deployment of multiple malware types is an obvious strategy to counter software diversity. Devices are assumed to automatically remove executable code, including unknown malware, and immediately download new diverse code from application stores on a semi-regular basis. The introduction of imperfect malware detection allows devices to also initiate unscheduled code removal and updates when infections are detected. In severe but rare cases, trained personnel must take a device offline to wipe its entire memory before installing the new software. The following model assumes that the self-repairing and diversity-enhancing approach removes all malware. Because it is hard to remove advanced malware, especially rootkits, from real systems, it is possible to adjust how often the model successful carries out code removals and updates.

10.1.1 Model Description

We model multimalware spreading over networked computing devices by a simple graph (no self-loops or parallel edges) with N nodes and a maximum of L node types for $L \ll N$. At time step $t = 0, 1, \ldots,$ the graph contains $D = D(t)$ of the L node types, where $D(0) = 1$. The D *active* node types represent classes of binary codes at the OS or application level of the devices' computing platforms; that is, nodes of the same type share an exploitable vulnerability while nodes of different types have no common exploitable vulnerabilities. The edges represent virtual communication lines. The number of active node types D measures the model's time-varying *diversity*.

Two nodes are neighbors if there is an edge between them. A node's degree k is the number of neighbors and $\langle k \rangle$ is the average degree over all nodes. All nodes change type with probability p at each time step to model the automated removal of executable code (including unknown malware), followed by immediate downloads of new diverse code from application stores. One of the L possible node types is selected with probability $1/L$, thus changing the initial monoculture into a polyculture with diversity L.

Initially, all nodes are susceptible to malware infections. There is one malware type per node type. All malware types have the same spreading mechanism. Whereas epidemiological models in the literature tend to model a single malware outbreak, we model systems with many outbreaks. One malware outbreak occurs with probability q at each time step. An outbreak initially infects a single susceptible node selected uniformly at random. A newly sick node infects all its susceptible neighbors of the same type during the next time step. Infected nodes change type with probability r at each time step to model the varying degree of imperfect malware detection followed by immediate malware removal and the installation of new diverse code. Any infected node becomes susceptible when it changes type.

It is possible to switch off automated malware detection by setting $r = 0$. We can also set $p = 0$ to disable automated software downloads. A small fraction of nodes can be immunized, that is, made resistant to malware infections. Immunized nodes do not change type or transmit infections to neighbors. As stated in Chap. 8, automated immunization or hardening includes the removal of non-essential software programs, the secure configuration of remaining programs, constant patching, and the use of firewalls and intrusion prevention systems. Other mitigation techniques, such as control-flow integrity [93], that induce code overhead and performance penalties can also be used on selected devices.

10.1.2 Model Limitations

As first observed in Chap. 8, it is hard to predict how malware will spread over a networked computing system because the propagation depends on the malware's spreading mechanism, the network topology, changing traffic loads, routing and filtering policies, the choice of communication protocols, and network failures and misconfigurations. Rather than trying to generate accurate spreading patterns under various network conditions, the model displays very fast worst-case malware spreading in which an infectious node immediately infects all its neighbors of the same type.

Although the model cannot predict spreading in a real networked system, it can demonstrate the usefulness of combining software diversity and imperfect malware detection to halt malware spreading. Compared to the model, actual malware is likely to spread slower, because the first attempt to infect a susceptible computing device will not always succeed and not all susceptible devices will be infected because some are unreachable in practice. Hence, it is reasonable to believe that modeled malware halting translates into halting in real systems.

We only study sparse spreading patterns with an average degree $\langle k \rangle$ much lower than the number of nodes N. Whereas nodes and edges can be deleted during a model run and new nodes and edges added to simulate changes in the malware's spreading mechanism, nodes cannot change position after they have been created.

10.2 Anti-fragility on Static Graphs

A worst-case spreading pattern given by a network with average degree $\langle k\rangle$ is homogeneous when all nodes have degrees $k \approx \langle k\rangle$. To keep the analysis manageable, we first study static spreading patterns represented by homogeneous networks and determine when the system model is anti-fragile to spreading.

Consider a homogeneous network with a single node type at time $t = 0, D(0) = 1$. If this monoculture is connected, then it is extremely fragile to malware spreading since a single sick node will infect all nodes as long as no node changes type. The model avoids fragile monocultures by allowing nodes to change type. Each time a node changes type, it selects a particular type with probability $1/L$. Consequently, the number of node types $D(t)$ will grow toward the maximum value L. The phase where $D(t)$ changes from one to L will be simulated later. Here, we assume that $D(t) = L$, where $t > t'$ for a small finite time t', and study the model after the fraction of infected nodes starts to fluctuate around a small time-averaged value f.

We need to determine an expression for the time-averaged fraction f of infected nodes. Let Q denote the set of susceptible nodes that are infected during T time steps. We first estimate the expected number of infected nodes in Q, denoted $E\{|Q|\}$. During each time step, there is a probability q that a single susceptible node is seeded with an infection. The probability that no neighboring node has the same type as this seed is approximately $(1 - 1/L)^{\langle k\rangle}$. If we choose a large diversity $L > \langle k\rangle$ such that this probability is large, then an infection will most likely spread at most from the seed to the nearest neighbors of the same type. Ignoring further spreading, each seed infects, on average, $\langle k\rangle/L < 1$ of its neighbors. Over T time steps, the expected number of seeds is $T \cdot q$ and about $T \cdot q \cdot \langle k\rangle/L$ susceptible neighbors will be infected, since the average fraction f of infected nodes is small. The expected number of susceptible nodes becoming infected during the period T is thus estimated by $E\{|Q|\} \approx T \cdot q \cdot (1 + \langle k\rangle/L)$.

Next, we determine the expected number of infected nodes that become susceptible during T time steps. All N nodes in a network change type with probability p to model periodic downloads of diverse software. Let P denote the set of infected nodes that change type (and become susceptible) due to periodic software downloads. We need to determine the expected size of P, denoted $E\{|P|\}$. The expected number of type changes over a period T is $p \cdot N \cdot T$. Since the fraction of infected nodes is f, the expected number of infected nodes that change type is $E\{|P|\} = f \cdot p \cdot N \cdot T$.

The remaining infected nodes at a time step detect their infections with probability r. Let R be the set of infected nodes that change type (and become susceptible) due to malware detection followed by an immediate software download. We also need to determine the expected size $E\{|R|\}$. The expected number of infected nodes is $f \cdot N \cdot T$ and the expected number of infected nodes changing type is $E\{|R|\} = r \cdot f \cdot N \cdot T$.

Over T time steps, the two sets P and R overlap. The total number of unique nodes changing type and becoming susceptible is given by the union $P \cup R$. The expected size is of this union is $E\{|P \cup R|\} = E\{|P|\} + E\{|R|\} - E\{|P \cap R|\} = pfNT + rfNT - prfNT = fNT(p + r - pr)$. The expected number of nodes changing type and becoming susceptible must be equal to the expected number of new infected nodes to maintain a stable time-averaged fraction f of infected nodes. Hence, the relation $E\{|P \cup R|\} = E\{|Q|\}$ results in the approximation

$$f \approx \frac{q\,[1 + \langle k \rangle / L]}{(p + r - pr)N} \tag{10.1}$$

for homogeneous networks with diversity $L > \langle k \rangle$.

Equation (10.1) shows that it is possible to maintain a small fraction f of infected nodes, even if malware detection is switched off ($r = 0$), by adjusting the software download probability p. Note that this property is based on the assumption that malware is removed during the software update process. Let p and r be small such that the value of pr is negligible compared to $p + r$. When malware detection is switched on ($r > 0$), the fraction of infected nodes reduces further for a fixed probability p. Hence, anti-fragile systems using imperfect detection and removal of new malware further reduce the fraction of infected devices, compared to robust systems that merely remove old code and download new diverse code periodically.

10.2.1 Simulations of Anti-fragility on Static Networks

To validate Eq. (10.1), we consider smartphones and other handheld computing devices that communicate via short-range Wi-Fi and Bluetooth links [85]. Infectious malware types can copy themselves to new devices by opening wireless connections. Malware can also propagate directly between Wi-Fi access points via wireless connections [94]. We represent the worst-case spreading patterns by homogeneous proximity networks. The system model was programmed in NetLogo [46] and generates a proximity network with average node degree $\langle k \rangle$ by first placing N nodes uniformly at random on a square. An edge is then added between a randomly chosen node and its closest neighbor in Euclidean distance. More edges are similarly added until the network has the desired average degree. Self-loops and multiple edges between nodes are not allowed.

Simulations were run on networks with $N = 5{,}000$ nodes, $L = 20$ node types, and different average degrees $\langle k \rangle$. The outbreak probability was $q = 10^{-2}$, the software download probability $p = 10^{-5}$, and the malware detection probability $r = 10^{-3}$. Table 10.1 lists the observed average, minimum, and maximum fraction of infected nodes over 100 runs, where each value was averaged over the last 10,000 time steps of a run. The table also reports an estimate of the average fraction obtained from Eq. (10.1). The good agreement between the simulated and calculated values shows that the expression can provide good estimates of the average fraction of infected nodes. Other model runs with different parameter values confirm the agreement between simulated and calculated values.

Table 10.1 The estimated average fractions of infected nodes in proximity networks with 5,000 nodes and increasing average node degree

Proximity networks				
Average degree	5	6	7	8
Estimated frac.	0.25 %	0.26 %	0.27 %	0.28 %
Simulated frac.	0.26 %	0.27 %	0.28 %	0.30 %
(max., min.)	(+0.09, −0.08)	(+0.08, −0.1)	(+0.11, −0.07)	(+0.16, −0.11)

The corresponding simulated fractions are averaged over 100 runs, with the largest observed deviations shown in parentheses

10.2.2 Anti-fragility on Large Static Networks

Since there is agreement between the average fractions of infected nodes obtained from the simulations and from Eq. (10.1), we use the equation to study anti-fragility to malware spreading on very large homogeneous networks. The required frequency of software download p and the frequency of malware detection r decrease as the size of a network grows, because $p + r$ is proportional to $1/N$. Hence, anti-fragility to malware occurs on large model networks for practical download and detection frequencies.

Consider a homogeneous network with 100 million nodes, that is, $N = 10^8$, and average degree $\langle k \rangle \ll L$. For outbreak frequency $q = 10^{-2}$ and an average fraction of infected nodes $f = 10^{-3}$, we have from Eq. (10.1) that $p + r \approx q/(fN) = 10^{-7}$. If each time step in the model is one second long, then there is a new malware outbreak every 100 seconds, on average. The fraction of infected nodes is maintained when the download frequency is $p \approx 10^{-7}$ without malware detection ($r = 0$), that is, each device has to download and install new software after about 116 days. If malware detection is added to our example, then the average fraction of infected devices is reduced. For $r = 10^{-4}$ and $p = 10^{-7}$, we have $f \approx 10^{-6}$. The calculations illustrate that anti-fragility to malware spreading scales to very large homogeneous networks.

10.3 Anti-fragility on Time-Varying Graphs

We now consider a modified system model with an unknown and time-varying worst-case spreading pattern that remains sparse over time. Even if the spreading mechanism varies, the spreading is mostly limited to the neighbors of the nodes seeded with infections as long as the spreading pattern remains homogeneous and the diversity remains much larger than the changing average degree.

We therefore study inhomogeneous spreading patterns containing a small fraction of nodes, called *hubs*, with degree k_h much larger than the time-varying average degree. A hub and its k_h neighbors form a star graph with the hub at the center. If all $k_h + 1$ nodes have a uniform distribution of L node types, then there are roughly

k_h/L neighbors of the same type as the hub. A susceptible hub is infected each time one of the neighbors of the same type is infected. A reinfected hub again infects all susceptible neighbors of the same type, ensuring a total of k_h/L infected neighbors.

Since a hub's k_h neighbors constitute a significant fraction of all nodes in a worst-case spreading pattern, we may very well have $k_h \gg L$. This is why hubs are referred to as super-spreaders, even in software polycultures. Note that malware does not need to be infectious for a hub to be a super-spreader; for example, a hub can be a popular website infected by malware that is inadvertently downloaded by many users accessing the site. While the non-infectious malware does not spread any further after the downloads, the number of infected devices k_h/L is still large.

Similarly, if the hubs' neighbors tend to have small degrees, most of the spreading of infectious malware will also be confined to the hubs' neighbors. Even when a hub regularly changes type, there will still be roughly k_h/L neighbors of the same type as long as all nodes have a uniform distribution of types. Over time, a hub that changes type will reinfect many neighbors as long as at least a few neighbors of different types are infected.

If hubs are connected in a small subnetwork and several hubs have the same type, then a large fraction of all nodes in a worst-case spreading network is infected very quickly. Hence, we need to "neutralize" hubs, especially tightly connected hubs, to make the actual spreading pattern more homogeneous such that, for any susceptible node of degree k, the expected number of neighbors of the same type is negligible ($k/L \ll 1$).

The following three malware simulations with different spreading patterns show that hubs can be immunized to gain anti-fragility to multimalware spreading. Rather than presenting plots averaged over many runs to obtain smooth curves, the figures plot single runs to better demonstrate how anti-fragile systems would actually behave.

10.3.1 Simulations of Anti-fragility

In the first simulation, the NetLogo model generates a time-varying spreading pattern. A model run starts with a proximity network with 2,000 nodes, average degree $\langle k \rangle = 4$, maximum degree 10, and diversity $D(0) = 1$. Initially, malware detection is turned off ($r = 0$). The fraction of infected nodes plotted in Fig. 10.1 reduces from about 91 to 8 % as the model changes from a fragile monoculture to a more robust polyculture with diversity $L = 5$. When malware detection is turned on ($r = 10^{-2}$), the fraction reduces further to 0.1 %. The plot confirms the advantage of introducing software diversity and applying imperfect malware detection.

Next, 75 % of the nodes and their adjacent edges are deleted and the network is regrown using the preferential attachment technique [95], with each new node connected to three existing nodes. The new nodes have a uniform distribution of node types. This simulated change in spreading mechanism causes the original homogeneous spreading pattern to change into an inhomogeneous spreading pattern. The new spreading pattern has an average degree $\langle k \rangle = 4.8$ and a maximum degree 33.

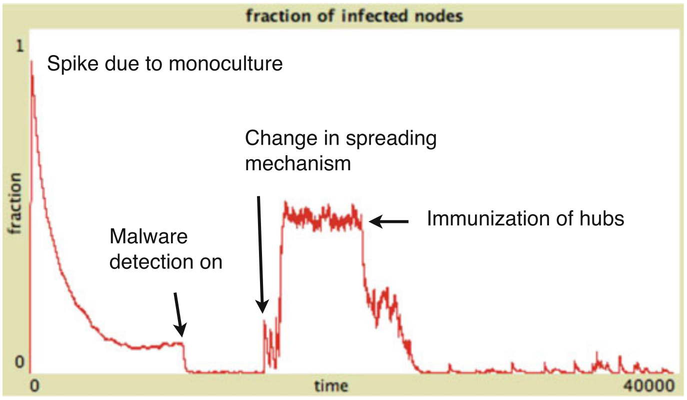

Fig. 10.1 The fraction of infected nodes in a changing network with 2,000 nodes, outbreak probability $q = 10^{-2}$, download probability $p = 10^{-3}$, and diversity $L = 5$. The plot illustrates the effects of increasing diversity, malware detection, a change in spreading mechanism, and hub immunization

The fraction of infected nodes increases to roughly 45 % because the diversity L is not large enough to prevent spreading from the new hubs, even though malware detection is still on. The sharp increase in the fraction of infected nodes illustrates that the malware halting is fragile to changes in the spreading pattern when the diversity is too small.

Finally, the 61 nodes with the largest degrees are immunized. When these hubs and their adjacent edges are ignored because they no longer contribute to malware spreading, the remaining spreading pattern has average degree $\langle k \rangle = 3.8$ and maximum degree 14. The fraction of infected nodes reduces to roughly 0.6 %, demonstrating the need to immunize super-spreaders in real networks to obtain more homogeneous spreading patterns.

In the second simulation, the NetLogo model starts with an inhomogeneous email network with 1,133 nodes, average degree $\langle k \rangle = 9.6$, maximum degree 71, and diversity $L = 8$. The largest hubs are immunized before the model run starts. As shown in Fig. 10.2, the fraction of infected nodes reduces to roughly 1 % as the monoculture turns into a polyculture. The model then erases 75 % of all nodes as before and creates an inhomogeneous network with 2,000 nodes. No new nodes are immunized and their types are uniformly distributed. The new subgraph of susceptible nodes has average degree $\langle k \rangle = 4.7$ and maximum degree 44. Unlike in the first simulation, there is no large change in the fraction of infected nodes in Fig. 10.2 because the diversity, the remaining immunized nodes, and the malware detection probability ($r = 10^{-2}$) together prevent significant spreading. The plot shows that the malware halting can be made robust to changes in the spreading pattern.

In the third simulation, the NetLogo model utilizes a static inhomogeneous spreading pattern with 10,670 nodes, 36 hubs that form a small connected subgraph, and many nodes with a low degree $k \approx \langle k \rangle = 4.1$. The hubs' degrees range from

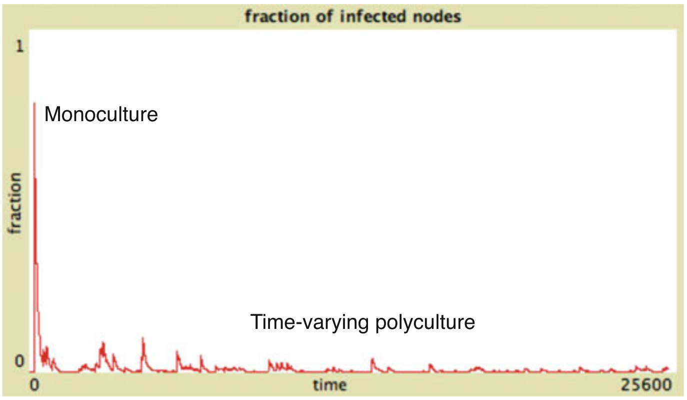

Fig. 10.2 The fraction of infected nodes in a time-varying e-mail network with outbreak probability $q = 10^{-2}$, download probability $p = 10^{-3}$, malware detection probability $r = 10^{-2}$, and diversity $L = 8$. Note that there is no visible change in the fraction when the spreading network changes

2,312 down to 102. In a real network, any immunization of hubs is likely to be imperfect because some infected hubs are not detected or because some hardened hubs still become infected. The model utilizes acquaintance immunization to simulate imperfect cloud-based detection and immunization of infected hubs [25]. This immunization technique chooses a set of nodes uniformly at random and immunizes one arbitrary neighbor per node. While the original set of nodes is unlikely to contain the few hubs in the network, the randomly selected neighbors are much more likely to be hubs, since many edges are adjacent to high-degree nodes.

Figure 10.3 plots the fraction of infected nodes. The fraction decreases as the diversity grows to $L = 14$ but stabilizes around 11 % because the hubs are not immunized. When acquaintance immunization selects 2 % of the nodes, all but three

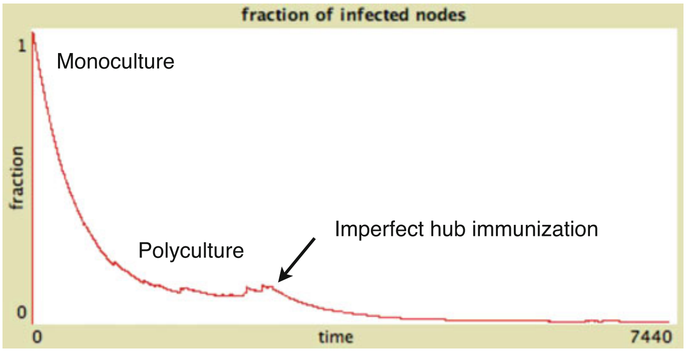

Fig. 10.3 The fraction of infected nodes in a static network with 10,670 nodes, outbreak probability $q = 10^{-1}$, download probability $p = 10^{-3}$, malware detection probability $r = 10^{-3}$, and diversity $L = 14$

of the hubs are immunized. The fraction of infected nodes reduces to about 0.1 %. Figure 10.3 illustrates that imperfect detection and immunization of hubs reduce the fraction of infected nodes, even when the spreading pattern contains a subnetwork of tightly connected hubs.

Additional simulations with varying parameter values confirm the model behavior reported. In particular, simulations using acquaintance immunization confirm the adequacy of imperfect hub immunization. The additional simulations further strengthen the claim that compiler-generated software diversity, periodic downloads of software from application stores, and imperfect malware detection/removal together provide a networked computing system with a degree of anti-fragility to multimalware spreading.

10.4 Discussion

The anti-fragile malware-halting technique scales to large networked systems because compiler-generated software diversity and malware detection can be implemented as cloud services. While empirical work is needed to determine the real-world performance of the combined services, it is encouraging that there exist commercial anti-malware solutions running in the cloud. According to Franz's research group [24, 89], it is cost-effective to compile diverse software in the cloud. Furthermore, the impact of software diversity on the runtime performance is small and it is possible to securely patch diverse software. Still, challenges remain.

While acquaintance immunization is useful for simulating imperfect detection and immunization of hubs, the strategy is not the best choice for real networked systems, because many potential super-spreaders, such as popular websites, are known. The challenge is to ensure that the owners of potential super-spreaders harden their systems. Users could be warned to stay away from infected websites, making it necessary for owners to remove the malware and harden the systems to get their users back. At the time of this writing, Google informs users and webmasters of unsafe websites (http://google.com/transparencyreport/safebrowsing).

Netflix's decision to induce failures in their production system to repeatedly increase robustness to downtime raises the question of whether to use infectious "goodware" to improve the detection of susceptible devices and speed up the learning process leading to anti-fragility to malware spreading. While ethical questions are associated with this approach, it is worth investigating.

Although users today regularly download software from application stores, more work is needed to create self-repairing (up to a point) devices that remove malware and install diverse software in a way acceptable to users.

What to learn from Part III

Part III analyzed how to prevent infectious malware from spreading over huge networks of computing devices. Through a series of analyses, we developed a malware-halting technique that stops frequent multimalware outbreaks with an unknown and time-varying spreading mechanism. The technique combines application stores with compiler-generated software diversity, imperfect malware detection, and the semi-periodic reinstallation of software on devices. If compiler-generated software diversity and malware detection are realized in the cloud, then the malware-halting technique scales to huge networks, because it does not require any tightly coupled interactions or adaptations between groups of computing devices.

More efficient malware-halting techniques exist that require less software diversity to halt malware outbreaks on spreading networks with known and unchanging topologies. The problem with these techniques is that the topologies of spreading networks are rarely known in practice. Furthermore, the topologies change over time as the malware writers change the spreading mechanisms. Finally, the previously known techniques require a high degree of central control, limiting the ability to scale to millions of devices. To ensure scaling, we relinquished central control and made a strategical decision to not let the perfect be the enemy of the good. Hence, instead of trying to minimize the needed software diversity, we focused on creating a simple technique that takes advantage of existing and proposed technologies to halt frequent multimalware outbreaks with unknown and changing spreading mechanisms.