Friends Troubleshooting Network: Towards Privacy-Preserving, Automatic Troubleshooting

Helen J. Wang[*], Yih-Chun Hu[†], Chun Yuan[*], Zheng Zhang[*], Yi-Min Wang[*]
[*]Microsoft Research
[†]U. C. Berkeley

***Abstract*— Content sharing is a popular usage of peer-to-peer systems for its inherent scalability and low cost of maintenance. In this paper, we leverage this nature of peer-to-peer systems to tackle a new problem: automatic misconfiguration troubleshooting. In this setting, machine configurations from the peers are "shared" to diagnose the misconfigurations on a sick machine. A key challenge for such a troubleshooting system is privacy preservation. To this end, we construct** *Friends Troubleshooting Network* **(FTN), a peer-to-peer overlay network, where the links between peer machines reflect the friendship of their owners. To preserve privacy, we use** *historyless and futureless random-walk* **in the FTN, during which search along with parameter aggregation are carried out for the purpose of troubleshooting. Many of our techniques can be applied to other application scenarios that require privacy-preserving distributed computing and information aggregation. We have also identified a number of open challenges that remain to be addressed.**

I. Introduction

Today's desktop PCs have not only brought to their users an enormous and ever-increasing number of features and services, but also an increasing amount of troubleshooting cost and productivity losses. Studies [14][15] have shown that technical support contributes 17% of the total cost of ownership of today's desktop PCs. A large amount of technical support time is spent on troubleshooting.

In this paper, we tackle an important troubleshooting category in which application failures are due to misconfigurations on the troubled machine. In our prior work [16], we developed an effective algorithm called *PeerPressure* for diagnosing such misconfigurations. PeerPressure uses the *common* configurations from a set of helper machines to identify the anomalous misconfigurations on the sick one. *Searching* for the right set of helper machines and *aggregating* helper configurations is the topic of this paper. Maintaining helper configurations at a centralized server or database is expensive as it requires continuous update and maintenance. Further, such a solution places complete trust on a single entity. Hence, the privacy of both helpers and troubleshooting users are of great concern. These reasons lead us to explore the peer-to-peer approach where peers share their content at low cost of maintenance and with inherent scalability. In our setting, the "content" refers to the machine configurations of the peers. Further, the trust is distributed among the peers. There are two essential goals in designing a PeerPressure-based peer-to-peer troubleshooting system:

- *Integrity*: We should preserve the integrity of the troubleshooting results.
- *Privacy*: We should protect privacy-sensitive configurations for both troubleshooting users and peer helpers during routing and information aggregation.

Ensuring *integrity* is challenging because malicious peers may lie about the applications they own and the configuration state they have, which can lead to incorrect troubleshooting results. A machine can be malicious either because its owner has ill intentions or because it is compromised by an attacker.

We cope with the ill-intentioned-user problem by designing well-established social trust into the troubleshooting framework. Today, when encountering computer problems, most people first seek help from their friends and neighbors. Based on this observation, we construct a *Friends Troubleshooting Network* (FTN), which is a peer-to-peer overlay network, where a link between two machines is due to the friendship of their owners. Here, we assume that a pair of friends intend to help each other out by contributing their own authentic, relevant non-privacy-compromising configuration information to each other for the purpose of troubleshooting. If the relevant configurations are privacy-sensitive, they will refuse to supply the information rather than giving false content. Further, just like in the real world, when Alice asks her friend Bob a question, if Bob only has a partial answer or does not know the answer, Bob can potentially ask his friend Carol *on Alice's behalf*, especially when Alice and Carol are not friends. This is exactly how a troubleshooting request recursively propagates in the FTN. One may quickly conclude that our system manifests *transitive trust*. However, in our example, because Carol and Alice are not friends, Carol may provide *untruthful* answers to Alice if Alice asks Carol *directly*. So, more precisely, FTN manifests *recursive trust* instead.

Coping with compromised FTN nodes is an open challenge. We provide a rough outline for integrity preservation under such conditions in Section VII-A.

Despite the friendship-based trust in the FTN, *privacy* remains a crucial goal in FTN since friends do maintain privacy from one another. In fact, much configuration state contains privacy-sensitive information, such as usernames, passwords, URLs visited, and applications installed. We achieve privacy through a *historyless and futureless random-walk* of an *ownerless* troubleshooting request, during which search as well as parameter aggregation are carried out for the purpose of PeerPressure troubleshooting. Unlike the traditional peer-to-peer search and routing protocols which are destination-driven, the historyless and futureless random-walk in FTN exhibits a new communication pattern which is *destination-free* searching and routing along with parameter aggregation.

For the rest of the paper, we first provide background on PeerPressure in Section II. We state our privacy objectives in Section III. In Section IV, we present our protocol design for privacy-preserving search and parameter aggregation. We discuss on how FTN achieves privacy objectives in Section V. Then we discuss the overhead of our protocol in Section VI. In Section VII, we list the open challenges remaining in this work. We compare and constrast our work with the related work in Section VIII and finally summarize in Section IX.

II. Background: PeerPressure

PeerPressure [16] assumes that an application functions correctly on most machines and hence that most machines have healthy configurations. It uses the statistics from a set of sample machines to identify anomalous misconfigurations. The distinct feature of PeerPressure in contrast with other work in this area [17] is that it eliminates the need of manually identifying a healthy machine as a reference point for comparison. We have experimented with the PeerPressure algorithm and a corresponding troubleshooting toolkit on Windows systems where most of configuration data is stored at a centralized registry. Figure 1 illustrates the operations of our PeerPressure troubleshooter.

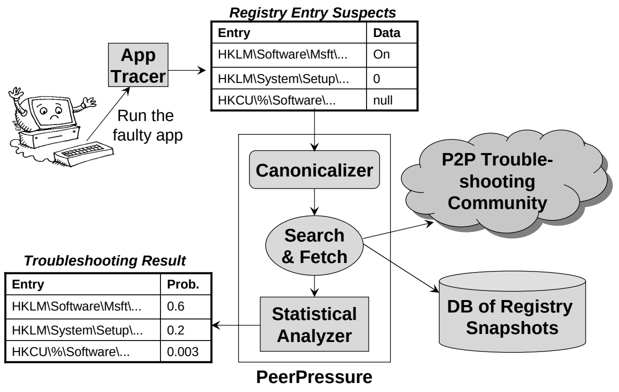

Registry Entry Suspects
Entry | Data
HKLM\Software\Msft\... | On
HKLM\System\Setup\... | 0
HKCU\%\Software\... | null
App Tracer
Run the faulty app
Canonicalizer
P2P Trouble-shooting Community
Search & Fetch
Troubleshooting Result
Entry | Prob.
HKLM\Software\Msft\... | 0.6
HKLM\System\Setup\... | 0.2
HKCU\%\Software\... | 0.003
Statistical Analyzer
DB of Registry Snapshots
PeerPressure

Fig. 1. PeerPressure Troubleshooter

PeerPressure first uses application tracing (with the "AppTracer") to capture the configuration entries and their values that are touched by the abnormal execution of the application under troubleshooting. These entries are misconfiguration *suspects*. Then, the canonicalizer turns any user- or machine-specific entries into a *canonicalized* form. For example, user names and machine names are all replaced with constant strings "USER_NAME" and "MACHINE_NAME", respectively. Next, from a sample set of helper machines, for each suspect entry, PeerPressure obtains the number of samples that match the value of the suspect entry, and the cardinality (the number of possible values this entry may have). PeerPressure uses these parameters along with the sample set size and the number of suspect entries to calculate the probability of a suspect entry being the cause of the symptom. The intuition behind this sick probability calculation is that the more conformant a suspect entry is with the samples, the more likely the entry is to be healthy. The top ranking entries with regard to the sick probability are diagnosed as the *root-cause candidates*. The sample set can be obtained either from a database of registry snapshots collected from a large number of user machines or from a peer-to-peer troubleshooting community such as the one described in this paper. We have demonstrated PeerPressure [16] as an effective troubleshooting method: Our PeerPressure troubleshooter can pinpoint the root-cause misconfiguration accurately for 12 out of 20 real-world troubleshooting cases and for the remaining cases, it can narrow down the root-cause candidates by three orders of magnitude.

III. Privacy Model and Objectives

Before we dive into our protocol design, we first state our privacy model and objectives.

A. *Private Information*

The information being communicated in FTN is PC configuration data. We denote the complete set of configuration data on a machine as D. A subset of D is identity-revealing, such as usernames and cookies, which we denote as D_i. The remaining set $D_r = D - D_i$ may contain information that compromises privacy when *linked* with user identity. Some examples of such information are URLs visited and applications installed. Our privacy objective is to protect *all* peers' privacy by *anonymizing* such privacy-sensitive information in D_r. Of course, this forbids D_i to be revealed.

B. *Attackers*

We assume a friendly operational environment where attackers are simply curious friends. We address compromised nodes in Section VII-A, and not else where.

C. *Attacks*

The ways attackers attempt to obtain privacy-sensitive information include the following:

1) Message inspection attack: Infer privacy-sensitive information passively by inspecting the messages that are passing by.
2) Polling attack: Repeatedly send fake troubleshooting requests to a friend to obtain and infer his private information.
3) Eavesdrop on machines on the same LAN.
4) Known topology attack: Discover the FTN topology through side channels. This information may be used to deduce privacy-sensitive information.
5) Gossip attack: Friends may gossip (i.e., collude) and correlate pieces of information.

IV. Privacy-Preserving Search and Parameter Aggregation Protocol in FTN

We assume that the FTN bootstraps and maintains its overlay links in the same way as Gnutella [8] or Kazaa [10] except that the neighbors are trusted friends' machines. We assume that each node and its immediate friends exchange their public keys through a secure out-of-band mechanism. The neighboring nodes use their public keys to establish secure channels for troubleshooting communications.

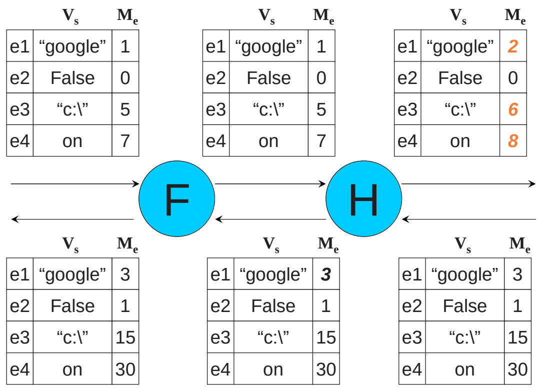
A. *Basic Approaches*

We take the following basic approaches to achieve our privacy objectives.

- **Integration of search and parameter aggregation in one transaction**: If the search is a separate step, which returns the IP addresses of helpers, then the querier can determine the applications running on the helpers' machines. Since application ownership could be private information, we integrate search and parameter gathering for PeerPressure into one step in such a way (next bullet point) that the parameter values at any point represents a collective state for a set of friends, and therefore does not reveal any individual state.
- **Historyless and futureless random-walk routing**: To preserve the privacy of the troubleshooting user as well as node owners on the path, we design the troubleshooting messages to be *ownerless*, and not to contain any routing history or future routing state such as the source and the nodes traversed or to be traversed. In addition, we make sure that the troubleshooting state gathered from the past is *aggregate* in nature so that individual state is disguised. Each node on the forwarding path of the random-walk is either a forwarder which simply proxies the request or a helper which contributes its own relevant configurations to the request and then proxies the request. Each node on the path keeps per-request state on the previous and next hop of the request. On the return path, the reply follows the same way back. The reply contains all the parameters needed by PeerPressure calculation at the sick machine.

B. *Protocol Details*

Figure 2 zooms into a segment of the random walk in FTN.



	V_s	M_e
e1	"google"	1
e2	False	0
e3	"c:\"	5
e4	on	7

	V_s	M_e
e1	"google"	1
e2	False	0
e3	"c:\"	5
e4	on	7

	V_s	M_e
e1	"google"	***2***
e2	False	0
e3	"c:\"	***6***
e4	on	***8***

	V_s	M_e
e1	"google"	3
e2	False	1
e3	"c:\"	15
e4	on	30

	V_s	M_e
e1	"google"	***3***
e2	False	1
e3	"c:\"	15
e4	on	30

	V_s	M_e
e1	"google"	3
e2	False	1
e3	"c:\"	15
e4	on	30

Fig. 2. FTN Search and Parameter Aggregation Protocol: in the forward path, H, the helper, updates value distribution (not shown) of each configuration entry e and the number of matches M_e if its value for e matches the sick value V_s. The forwarder only proxies the request. On the return trip, the complete PeerPressure parameter information is passed back to the sick machine following the same path. (The role of being a helper or a forwarder is only known to the node itself, and not anyone else.)

1	$H(app.exe)$ where H is a one-way hash function.
2	Remaining number of samples needed, R
3	For each suspect entry e, EntryName, SickEntryValue, the number of matches M_e, and the value distribution of e from the past samples

TABLE I
FTN REQUEST

1) **Creating a Request on the Sick Machine***:* A sick machine first filters out the identity-revealing entries from the suspects, for example by removing all entries that contain usernames. Then it creates a troubleshooting request as shown in Table I.

The entry value distributions in field 3 are needed by the sick machine for determining the most popular values as legal values for root cause candidates as well as determining the cardinality of the entry (Section II). The size of the value distribution depends on the cardinality of the entry. According to our study in [16], 87% of Windows registry entries have a cardinality of 1 and 94% have no more than 2. So, value distributions do not add significant overhead to our messaging.

To preserve anonymity, the requester initializes the value distribution with random values.

We identify each request with $RqID = H(n, AllEntryNames)$, where n is a nonce generated at the sick machine, and H is a one-way hash function.

2) **Forward Path and Parameter Aggregation***:* The sick machine establishes a secure channel with an available friend chosen at random, then sends it the troubleshooting request. The friend sends an *ACK* if it can become either a forwarder or a helper for the request. If no *ACK* is received upon timeout, then the requester tries another friend chosen at random. To avoid routing loops, if a friend has seen the *RqID* of an arriving request in the past, the friend replies with a *NACK*.

If a node is involved in forwarding or helping and if it is capable to help because it also runs the application under troubleshooting, then the node becomes a helper with the probability of P_h. Without P_h, the second-to-last-hop node can potentially infer information about the last-hop-node. When the application under troubleshooting is very popular, with high probability, the last-hop node is capable to help. Then the second-to-last-hop node can correlate request and reply to infer the last-hop helper's configuration state. Nonetheless, even with P_h, a node can poll its next hop using fake requests with $R = 1$ to infer information statistically. We make such attacks more difficult with a bimodal P_h where P_h takes a smaller value for requests with small R's.

A helper needs to update the troubleshooting request accordingly. It increments M_e, the number of matches for each suspect entry e, when the helper's value matches that of the sick's. And the helper updates the value distributions in the request based on its own respective entry values. (With our trust model, uncompromised FTN nodes do not lie about these values (Section I); and we address compromised nodes in Section VII-A). Then, the helper decrements R. If R is positive, the helper proxies the request to one of its friends.

Each node on the forwarding path must record the *RqID*, the request arrival time, the previous and next hop friend along with R. There is timeout associated with each request.

3) **Last Hop and Return Path***:* If R becomes 0, the node is on the last hop. The last-hop node waits for a random amount of time, then sends the reply back to the previous hop. Without the random wait, the second-to-last hop node could know that the reply comes from the last hop, then correlate the M_e's in request and reply to infer the last-hop node's values. It is possible that a curious friend launches a polling attack (Section III) with $R = 1$ and conducts statistical timing analysis to infer its next hop's private information. To make such statistical attacks more difficult, the random wait can be uniformly drawn from a large range, e.g., 15 hops.

The reply follows the request path back to the sick machine. The sick machine first subtracts the random initialization from the value distributions. Then it performs PeerPressure diagnosis.

V. Achieving Privacy Objectives

The historyless and futureless random-walk along with integrated search and parameter aggregation counters the message inspection attacks and polling attacks (see Section III-C). The use of secure channel on each overlay link counters the eavesdropping attack. The use of P_h makes the potential inference in known topology attack difficult. We can mitigate most gossip attacks with the use of P_h along with historyless and futureless routing, except in one scenario: If a victim has two gossiping friends, whenever the victim "helps" with troubleshooting and is en route between the two, its information could be inferred by the two gossipers. A potential technique for reducing the impact of this attack is random perturbation which is discussed in Section VII-B.

VI. Response Time and Bandwidth Overhead

In FTN, the troubleshooting response time is dictated by the number of hops a troubleshooting request and its reply traverse, which is $N/P_h * 2$ where N is the number of samples needed. When $N = 10$, PeerPressure is already effective; nonetheless, the larger the N is, the better the root-cause ranking becomes in general [16]. The response time can be improved with fan-outs and by gathering a fraction of samples on each branch.

In terms of bandwidth overhead, for the Windows applications we evaluated in [16], there is a median of 1171 suspect entries. So, according to Table I, our troubleshooting messages are about 80 KB.

VII. Open Challenges

A. Integrity in the Face of Compromised FTN Nodes

Though friends may be trustworthy, their computers' behaviors do not necessarily carry out their owners' intentions when they are compromised by attackers. To defend integrity when some FTN nodes may be compromised, all nodes on the request path remember the troubleshooting request (see Table I) that they forward. Then, on the return trip, each node checks these invariants: the additional number of matches for each suspect does not exceed R, the recorded remaining number of samples needed; for each suspect, the count in value distributions in the reply did not decrease; and the sum of all counts in the reply minus the sum of all counts in the request equals to R. Only when these invariants are maintained, a node forwards on the reply. Ensuring absolute integrity here remains an open challenge.

Another serious attack is that *one* compromised node can bring in his gang of malicious friends which could all be himself in the form of the Sybil attack [5]. A large majority of "Sybils" impact the troubleshooting result without violating any integrity invariants mentioned above. One way to counter this attack is to initiate multiple troubleshooting requests, hoping that a majority of them will return with correct troubleshooting results, and therefore weed out the incorrect ones that have compromised nodes on the path. The earlier a compromised node appears on a path (i.e., the higher the R is), the more room the compromised node has to sway the final troubleshooting result. In such cases, using a high fan-out (Section VI) reduces R, and limits the compromised node's influence. Further analysis is needed on the number of requests needed given the positions of the compromised nodes.

B. Random Perturbation for Gossip Attack

In a gossip attack (Section III), two common friends can collude to infer a third friend's private information by noticing the changes to M_e. To reduce the impact of this attack, it might be feasible for each node on the path to apply random perturbation [1] to R, the remaining number of samples required, M_e, the number of matches, and value distribution updates. For example, instead of incrementing M_e for Suspect e when a helper has a matching value, the helper adds a random noise to M_e to confuse the gossipers. Nonetheless, it is unclear how such random noise affect the final troubleshooting result especially when the total number of samples is small. We are investigating the applicability of this technique.

C. Ownerless Troubleshooting Request

Removing identity-revealing information from the original troubleshooting request is non-trivial since entries that are not identity-revealing individually could be combined to identify a user. Systematic recognition of all identity-revealing vectors of configuration entries is an open challenge.

VIII. Related Work

There is much related work in the arena of anonymization. Our chain style of historyless and futureless random-walk is similar in spirit to that of FreeNet [4] and Crowds [13]. FreeNet is a distributed anonymous information storage and retrieval system. Crowds is for anonymous web transactions. Chaum's approach to anonymization is based on the use of *mixes* [3] which serve as proxies to provide sender-receiver unlinkability through traffic mixing. Onion routing [9] extends the mixes with layers of onion-style pre-encryptions. Tarzan [6] implements the mix idea using a peer-to-peer overlay and provides sender anonymity and robustness to the mix entry point.

All of the above anonymization techniques address point-to-point communications. However, our protocol in FTN takes the form of one-to-many communications and is destination-free, since a sample set is drawn from the peers. As a result, nodes on the path must look into the packet content and potentially modify it before proxying it forward. Hence, our application additionaly requires user privacy not being compromised in this process of state aggregation.

Our problem of privacy-perserving parameter aggregation shares much similarity to the problem of secure and privacy-preserving voting [7][2] with four distinctions. First, voting requires voters to be authenticated by a centralized authority, such as the government. Second, our protocol requires participation privacy additionally; otherwise, the privacy of the application ownership is compromised. Third, unlike some voting scenarios where there are limited number of voting candidates, our problem scenario does not have such a limit. Lastly, while voting requires precise vote tallies, PeerPressure uses the statistics which allows small amount of inaccuracies; therefore, random perturbation could potentially be used for privacy in our scenario.

The authors of SIA[12] presented a set of techniques for secure information aggregation in sensor networks with the presence of malicious sensors and aggregators. The integrity of information aggregation is achieved essentially through authentication which is identity-revealing. In FTN, we cannot do the same because of the privacy concerns.

ACME [11] is a scalable, flexible infrastructure for monitoring, analyzing, and controlling Internet-scale systems. ACME collects, aggregates and reduces nodes' health data as the data is routed through a peer-to-peer network, but it does not address privacy or integrity.

IX. Concluding Remarks

In this paper, we introduce automatic troubleshooting as an interesting application that can also benefit from the content-sharing nature of peer-to-peer systems while being legal at the same time. We leverage an automatic troubleshooting algorithm, PeerPressure [16], which uses statistics among the peers to diagnose the anomalous misconfigurations on a sick machine. For a privacy-preserving PeerPressure-based peer-to-peer system, we construct a Friends Troubleshooting Network (FTN) which is a peer-to-peer overlay network where the links between peer machines reflect the friendship of their owners. The FTN nodes manifest *recursive trust* rather than transitive trust. In FTN, we use *historyless* and *futureless* random-walk for integrated search and parameter aggregation for PeerPressure. We believe these techniques can be applied to other application scenarios that require privacy-preserving distributed computing and information aggregation. There is much future work ahead of us in making FTN a reality. Integrity preservation in the face of compromised nodes, the feasibility of random perturbation in this setting, and recognizing identity-revealing configuration entries remain open challenges.

X. Acknowledgement

Josh Benaloh, John Docceur, John Dunagan, Jon Howell, David Oppenheimer and Alf Zugenmaier have given us invaluable discussions and critiques for numerous drafts of this paper from the technical content to the presentation. We are grateful to their generous help. We also thank the anonymous reviewers for their insightful comments and suggestions.

References

[1] Agrawal, R., and Srikant, R. Privacy Perserving Data Mining. In *Proceedings of SIGMOD* (2000).

[2] Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, Sept. 1987.

[3] Chaum, D. L. Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms. In *CACM* (1981).

[4] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. In Proc. International Workshop on Design Issues in Anonymity and Unobservability. In *Proceedings of International Workshop on Design Issues in Anonymity and Unobservability* (2001). Lecture Notes Computer Science Volume 2009.

[5] Douceur, J. R. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (2002).

[6] Freedman, M. J., Sit, E., Gates, J., and Morris, R. Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer. In *IPTPS* (2002).

[7] Fujioka, T., Okamoto, T., and Ohta, K. A Practical Secret Voting Scheme for Large Scale Elections. In *Proceedings of Auscrypt* (Dec. 1992).

[8] The Gnutella v0.6 Protocol, Gnutella Development Forum, 2001.

[9] Goldschlag, D. M., Reed, M. G., and Syverson, P. F. Onion Routing for Anonymous and Private Internet Connections. In *CACM* (Feb 1999).

[10] KaZaa. http://www.kazaa.com.

[11] Oppenheimer, D., Vatkovskiy, V., Weatherspoon, H., Lee, J., Patterson, D. A., , and Kubiatowicz, J. Monitoring, Analyzing, and Controlling Internet-Scale Systems with ACME. Tech. Rep. UCB/CSD-03-1276, U. C. Berkeley, October 2003.

[12] Przydatek, B., Song, D., and Perrig, A. SIA: Secure Information Aggregation in Sensor Networks. In *Proceedings of ACM SenSys* (Nov 2003).

[13] Reiter, M. K., and Rubin, A. D. Crowds: Anonymity for Web Transactions. In *ACM Transactions on Information and System Security* (Nov 1998).

[14] Silver, M., and Fiering, L. Desktop and Notebook TCO Updated for the 21st Century, September 2003.

[15] Web-to-Host: Reducing the Total Cost of Ownership, The Tolly Group, May 2000.

[16] Wang, H. J., Chen, Y., Platt, J., Zhang, R., and Wang, Y. M. PeerPressure, A Statistical Method towards Automatic Troubleshooting. Tech. Rep. MSR-TR-2003-80, Microsoft Research, Redmond, WA, Nov 2003.

[17] Wang, Y.-M., Verbowski, C., Dunagan, J., Chen, Y., Wang, H. J., Yuan, C., and Zhang, Z. STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support. In *Proceedings of LISA* (2003).