Online Accumulation: Reconstruction of Worm Propagation Path[⋆]

Yang Xiang, Qiang Li, and Dong Guo

College of Computer Science and Technology, JiLin University
ChangChun, JiLin 130012, China
sharang@yahoo.cn, li_qiang@jlu.edu.cn, guodong@jlu.edu.cn

Abstract. Knowledge of the worm origin is necessary to forensic analysis, and knowledge of the initial causal flows supports diagnosis of how network defenses were breached. Fast and accurate online tracing network worm during its propagation, help to detect worm origin and the earliest infected nodes, and is essential for large-scale worm containment. This paper introduces the Accumulation Algorithm which can efficiently tracing worm origin and the initial propagation paths, and presents an improved online Accumulation Algorithm using sliding detection windows. We also analyzes and verifies their detection accuracy and containment efficacy through simulation experiments in large scale network. Results indicate that the online Accumulation Algorithm can accurately tracing worms and efficiently containing their propagation in an approximately real-time manner.

keywords: Worm, Propagation path, Online tracing, Containment

1 Introduction

Network worms allow attackers to control thousands of hosts in a short time, launch DDoS attacks, steal security information, and destroy critical data. Since 2001, Slammer and other network worms[1, 2] have brought unprecedented threat and damage to the Internet. There is increasing threat of network worms against computer system security and network security.

Tracing worm's attack paths (i.e., obtaining the propagation paths of network worm) [3, 5, 6] can dig out the initial victims and the infect sequence of hosts. Even if only partial path can be obtained, it still has significance in worm containment, evidence collecting and investigating.

Worm containment works by detecting that a worm is operating in the network and then blocking the infected machines from contacting further hosts. A key problem in containment of scanning worms is efficiently detecting and suppressing the scanning. Since containment blocks suspicious machines, it is critical that the false positive rate be very low[14].

In addition, the overwhelming majority of the attack traffic originates from victims of the attack, as opposed to the true source of the attack. While network terminals deploy corresponding defense gradually, the infected hosts may no longer participate

[⋆] Supported by NSFC (60703023). Corresponding author: Qiang Li, li_qiang@jlu.edu.cn

in the following attack. In contrast, worm source and the initial infected hosts may be artificially controlled, these hosts are more danger, and can not be easily detected or recovered. So, reconstruct worm source and the initial causal flows, makes worm containment more effective.

However, the reaction time of efficient worm containment could be less than a few hours or even minutes[12]. For example, Code-Red II worms infected more than 359,000 computers on the Internet in less than 14 hours[2]. Slammer worms probed all four billion IPv4 Internet addresses for potential victims in less than 10 minutes[1]. Therefore, it is necessary to do research on online worm tracing approaches in complex network environments, to trace network worm origins in a real-time manner.

In order to achieve online tracing, the following issues must be resolved: (1) shorten the time required for reconstructing worm propagation paths in order to reduce computation complexity; and (2) guarantee reconstruction of paths continuously.

Contribution of this paper includes: (1) introduction of the *Accumulation Algorithm* for reconstructing worm propagation paths, which can fleetly and efficiently trace worm attacking origins and initial propagation paths; (2) introduction of the online *Accumulation Algorithm* using sliding windows, which can obtain worm origin and tracing initial attack edges at the early days of worm propagating; and (3) deployment of a simulation environment for worm propagation in large scale network, verify the performance of our algorithm.

This paper is organized as follows. Section 2 introduces the related work of worm detection and containment; section 3 gives some definitions and assumptions of the following analysis; section 4 proposes the *Accumulation Algorithm* and prove its feasibility through theoretical analysis; section 5 introduces the online *Accumulation Algorithm*; section 6 verify the accuracy and efficiency of our algorithm through simulation experiments in large scale network; finally section 7 gives a conclusion.

2 Related Work

Worm containment has been studied in previous work. Network-based worm containment techniques can be classified into two major categories, that is, address blacklisting and signature-based filtering. Besides network-based techniques, Vigilante et al.[15] employs the collaboration among end hosts to contain worms by using self-certified alerts. Shield et al.[16] installs host-based network filters that are vulnerability specific and exploit generic once a vulnerability is discovered and before a patch is applied. DOMINO et al.[7] builds an overlay network among active-sink nodes to distribute alert information by hashing the source IP addresses. Worminator et al.[13] summarizes portscan alerts in Bloom filters and disseminates them among collaborating peers. Our work is focus on online tracing worm origin and initial propagation paths, help to quickly and effectively deploy worm containment.

To date, merely a few approaches for offline tracing the sequence of hosts infected by a worm are proposed. Xie et al.[5, 11] offered a randomized approach that traces the origin of a worm attack by performing a random walk over the hosts contact graph, which is generated by collecting flow rates between potential victims during the worm's propagation. Besides, aiming at the flow characteristics of mobile worm in wireless networks, Sarat et al.[9] improved random moonwalk algorithm so that the algorithm tends to be effective continuously. Rajab et al.[3] presented a simple technique that uses the history data acquired through a network telescope to infer the actual sequence of host infections. A different approach was proposed by Kumar et al.[6] where a Witty worm was reversely engineered to recover the random scanning algorithm and corresponding initial seeds. Finally, using protocol graph, Collins et al.[8] detect hit-list worm and identify attack origin through monitoring the abnormal changes in various of protocol graphs.

3 Problem Formulation

We consult some definitions in [5]: the host communications in network is defined as a directed graph $G =< V, E >$, called *host contact graph*. Nodes of G is a tuple set $V = H \times T$, where H is all hosts in the network and T is time. The set of edges E is a subset of $V \times V$. Each direct edge $e =< u, t^s, v, t^e >$ in the *host contact graph* represents a network flow, where $< u, t^s >\in H \times T$ represents source host and start time, $< v, t^e >\in H \times T$ represents destination host and finish time. An edge is defined as an *attack edge* if it carries attack traffic, whether or not it is successful in infecting the destination host. An *attack edge* is defined as a *causal edge* if it corresponds to a flow that successfully infects a normal host. All other edges in G besides *attack edges* are called *normal edge*.

If two edges $e_1 =< u_1, t_1^s, v_1, t_1^e >$, $e_2 =< u_2, t_2^s, v_2, t_2^e >$ in G satisfy the condition $u_2 = v_1$ and $t_1^e < t_2^s < t_1^e + \varDelta t$ ($\varDelta t$ is a pre-determined time interval parameters), then e_2 is called e_1's *successor*, e_1 is called e_2's *precursor*. All e's *precursors* represent as: e_{pre}^1, $\cdots$, e_{pre}^j, $\cdots$, $e_{pre}^{PRE(e)}$, $PRE(e)$ is the total number of e's *precursor*. Similarly, all e's *successors* represent as: e_{suc}^1, $\cdots$, e_{suc}^j, $\cdots$, $e_{suc}^{SUC(e)}$, $SUC(e)$ is the total number of e's *successor*. *Precursor* and *successor* describe the relationships between the neighbor edges.

Under normal circumstances, we assume that there is only one worm origin in the network, so the worm's propagation process forms a tree (defined as *causal tree*). A path in *causal tree* from the root to one of the leaves called a *causal chain*. *Causal tree* is formed by all *causal edges* in G. The root of *causal tree* denotes the worm attack origin, while *causal edges* from levels higher up in the *causal tree* denote the initial attack sequences. Reconstructing worm origin and the initial attack sequences has significance in restraining evolution of worm in investigating and collecting evidence. After we know *host contact graph* G, our algorithm identifies a set of edges that, with high probability, are edges from the top levels of the *causal tree* (i.e., initial attack sequences after worm breaks out).

4 Accumulation Algorithm

Network worms can infect a large number of hosts in a very short period of time. This requires worm tracing algorithms be able to obtain propagation path as soon as possible in order to reduce loss. At the same time, traffic data in the network are generated very fast, usually occupying a large portion of bandwidth. Consequently, it requires that the time and space complexity of the algorithm to be near-linear. We use dynamic programming to optimize the implementation of our *Accumulation Algorithm*. With even millions of input size, the algorithm is able to complete in a very short time, implying more time for the deployment of defense against possible future attacks for the same worm.

4.1 Algorithm Specification

In order to continuously infect other hosts, after a host has been infected, it usually sends more flows compare to former, while there is no significant increase in the number of received flows[5]. Compare to a *normal edge*, a *causal edge* has more *successors* while the number of *precursor* is similar on average. Motivated on this difference, we propose a worm propagation path reconstruction method - *Accumulation Algorithm*. First of all, we assign each edge with the same weight; then after K iterations of weight's 'aggregation - cumulation' (*accumulation process*), more weights tend to aggregate to *causal edges*; finally we pick out top Z edges (*TOP-Z*) which have the largest weight to trace initial propagation paths and reconstruct top levels of *causal tree*.

We define $p(e, i)$ as the *weight increment* in the i-th *accumulation process*, then when the algorithm is complete e has its total weight value $p(e) = \sum_{i=1}^{K} p(e, i)$. In fact each *accumulation process* is a redistribution of the previous *weight increment*. Specifically, each accumulation process evenly distributes the previous *weight increment* $p(e, i-1)$ to e's every *precursors* $e^1_{pre}, \cdots, e^j_{pre}, \cdots, e^{PRE(e)}_{pre}$, counting as a faction of the current weight increment for each of the precursors. After K iterations, each edge's *weight increment* is continuously distributed to their *precursors*. In fact, the redistribution process of *weight increment* is a weight accumulates process performed along the reverse *causal chain*. The following snapshot illustrates the *Accumulation Algorithm*:

```
STEP 1: $i = 0$; $p(e) = 0.0$; $p(e, 0) = 1.0$;
STEP 2: $i = i + 1$;
```

$$p(e, i) = \sum_{j=1}^{SUC(e)} \frac{p(e^j_{suc}, i-1)}{PRE(e^j_{suc})}; \tag{1}$$

```
        $p(e) = p(e) + p(e, i)$;
STEP 3: If $i \leq K$ goto STEP 2, else goto STEP 4;
STEP 4: Pick out TOP-Z, Reconstruct top levels of causal tree.
```

Adjust of the *weight increment* can be treated as a redistribution process, no additional weight is generated. The redistribution process of *weight increment* is called 'aggregation', and adding one's *weight increment* to its total weight is called 'cumulation'. In the *accumulation process*, a weight value aggregates to the top levels of *causal tree* along reverse *causal chain*. This becomes our primary evidence used to discover the initial attack sequences.

During the K iterations, one can completely generates the $(i + 1)$-th weight increment according to the i-th value. That is, the 'future' weight increment relies on only the 'current' weight, instead of the 'past' weight. So we can use dynamic programming method gradually calculate all the $p(e, i)$ ($e \in E, 1 <= i <= K$). Therefore, the time and space complexity of weight redistribution is $O(|E|)$. *Accumulation Algorithm* has K iterations of weight redistribution, thus the total time complexity is $O(K \times |E|)$. Experiments show that ideal accuracy can be achieved even if K is very small. Therefore, it is roughly the case that *Accumulation Algorithm* has a linear time complexity and space complexity.

4.2 Analysis and Prove

Accumulation Algorithm tries to identify initial *causal edges* with high accuracy and reconstruct the *causal tree*. To illustrate the accuracy and feasibility of our algorithm, we need to model the traffic data and worm attack. Suppose a normal host sends A flows in Δt seconds, and an infected host sends B flows in Δt seconds, including A normal flows and $B - A$ attack flows (clearly there is $B > A$). For an aggressive worm, the number of sending flows increase significantly after a host is infected (i.e., $B >> A$). But before and after infection, the number of flows a host received in Δt seconds remains almost unchanged (defined as C).

We define an edge $e =< u, t^s, v, t^e >$ as a *malicious-destination edge* if host v is infected at (or before) time t^e, marked as e_m. Other edges is called *normal-destination edge*, marked as e_n. Every *normal-destination edge* is a *normal edge*, but *malicious-destination edges* include all *causal edges*, some *normal edges* and a part of non-causal *attack edges* (v has been infected before time t^e). Assume that, on average, x *normal edges* sent by a host in Δt seconds are *malicious-destination edges*.

Next we prove that $p(e_m, i) > p(e_n, i)$ for all $1 <= i <= K, e_m, e_n \in E$.

Proof. Using Mathematical Induction:

1. First, prove that $p(e_m, 1) > p(e_n, 1)$:

$$p(e_m, 1) = \sum_{j=1}^{SUC(e_m)} \frac{p((e_m)^j_{suc}, 0)}{PRE((e_m)^j_{suc})} \approx \frac{B}{C}, \quad p(e_n, 1) = \sum_{j=1}^{SUC(e_n)} \frac{p((e_n)^j_{suc}, 0)}{PRE((e_n)^j_{suc})} \approx \frac{A}{C},$$

$$\because p(e_m, 1) - p(e_n, 1) \approx \frac{B-A}{C} > 0, \quad \therefore p(e_m, 1) > p(e_n, 1).$$

2. Assume $p(e_m, i) > p(e_n, i)$ when $0 < i < K$, prove that $p(e_m, i + 1) > p(e_n, i + 1)$:

$$p(e_m, i + 1) = \sum_{j=1}^{SUC(e_m)} \frac{p((e_m)^j_{suc}, i)}{PRE((e_m)^j_{suc})} \approx \frac{A-x}{C} \cdot \overline{p(e_n, i)} + \frac{B-A+x}{C} \cdot \overline{p(e_m, i)},$$

$$p(e_n, i + 1) = \sum_{j=1}^{SUC(e_n)} \frac{p((e_n)^j_{suc}, i)}{PRE((e_n)^j_{suc})} \approx \frac{A-x}{C} \cdot \overline{p(e_n, i)} + \frac{x}{C} \cdot \overline{p(e_m, i)},$$

Thereinto $\overline{p(e_m, i)} = \frac{\sum_{e \in E} p(e_m, i)}{\sum_{e \in E} 1}, \quad \overline{p(e_n, i)} = \frac{\sum_{e \in E} p(e_n, i)}{\sum_{e \in E} 1}.$

$$\because p(e_m, i + 1) - p(e_n, i + 1) \approx \frac{B-A}{C} \cdot \overline{p(e_m, i)} > 0, \quad \therefore p(e_m, i + 1) > p(e_n, i + 1).$$

So we have proved that $p(e_m, i) > p(e_n, i)$ for all $1 <= i <= K, e_m, e_n \in E$.

From the above proof, we can get that $p(e_m, i) - p(e_n, i)$ is proportional to $B - A$. For aggressive worm $p(e_m, i) >> p(e_n, i)$ because of $B >> A$. While the *accumulation process* is proceeding, $p(e_m, i) - p(e_n, i)$ increases gradually, and the weight advantage of *malicious-destination edges* grows. Furthermore, because the *accumulation process* follows the reverse order of *causal chain*, initial *causal edge* will get more weight. Thus these early propagation paths can be highlighted from all the edges.

However, *malicious-destination edges* include not only *causal edge*, but also some of the *normal edge* and non-causal *attack edge*. In the early phase of worm propagation, there is not much infected host, so the vast majority of *malicious-destination edges* are *causal edges*. In the late phase, almost all the vulnerable hosts in the network have been infected – the result being that the amount of *normal edges* and non-causal *attack edges* in all *malicious-destination edge* increase. Therefore, tracing worm in the early phase has a lower false negative rate, and thus is helpful for detecting worm as soon as possible, saving more time for defense against continues spread of the worm.

5 Online Accumulation Algorithm

With online tracing, propagation paths can be detected in the initial phase (e.g. 30 minutes) after the worm breaks out. As a result, inhibition and defense can be launched in the earlier stage, reducing some loss otherwise.

In the related works, we mention that enabling a detection algorithm to execute in real-time usually exploits sliding windows [4, 17]. The *Accumulation Algorithm* is able to promptly obtain the detection results and thus provides an ameliorate condition for online tracing. Based on the offline *Accumulation Algorithm*, we propose an online tracing algorithm also based on sliding window, as follow:

```
STEP 1. $i = 0$;
STEP 2. Collect traffic date within recent S seconds, construct the host contact graph $G_i$ using these traffic data;
STEP 3. Execute the Accumulation Algorithm in $G_i$, obtain top-$z_i$;
STEP 4. Compose TOP-$Z_i$ by extracting Z edges with the highest weight values from top-$z_i \bigcup$ TOP-$Z_{i-1}$. Reconstruct current causal tree via TOP-$Z_i$;
STEP 5. $i = i + 1$, iterate again from STEP 2 after R seconds.
```

This algorithm has many advantages: First, it is triggered every R seconds, thus worm can be detected as soon as possible. Second, each run needs to collect traffic data only within the recent S seconds – a large amount of overhead is avoided and improves its efficiency. One disadvantage of this algorithm is relatively low detection accuracy cased by the fact that it relies on only partial data.

6 Simulation Experiments

This section is constructed as follows. Section 6.1 gives the performance metrics; section 6.2 figure out our simulation methodology; section 6.3 discuss the parameters' influence on the performance of *Accumulation Algorithm*; section 6.4 discuss the parameters' influence on the performance of the online *Accumulation Algorithm*; section 6.5 illustrate effect of worm containment by the online *Accumulation Algorithm*.

6.1 Evaluation Methodology

To quantitatively evaluate the performance of *Accumulation Algorithm*, first we consider the following two metrics:

Attack edge Accuracy (AA) = $\frac{\text{\# attack edge in TOP-Z}}{Z}$;

Causal edge Accuracy (CA) = $\frac{\text{\# causal edge in TOP-Z}}{Z}$;

Further more, *Accumulation Algorithm* is designed to identify worms initial attack sequences. In the following experiments the earliest 10% *causal edges* (defined as *INIT-10%*) are considered as 'initial attack sequence'. Then there are two more metrics to evaluate the ability of tracing initial causal edges:

False Negative (FN) = $\frac{\text{\# edge in INIT-10\% but not in TOP-Z}}{\text{\# causal edge}}$;

False Positive (FP) = $\frac{\text{\# edge in TOP-Z but not in INIT-10\%}}{\text{\# non-causal edge}}$;

6.2 Simulation Methodology

In the worm detection works, experimental data usually produced by mixing real-world network traffic and man-made worm propagation flows [5, 8, 9]. Using pre-captured real network flows as the background traffic makes repeating experiments more convenient. Background flows often captured from main switches or routers. Worm attack flows are added artificially base on the real-world background data.

We use a part of NZIX II[10] trace from WAND as our background traffic data. This is a 9000 seconds long GPS-synchronized IP header traces captured at the New Zealand Internet Exchange, including exchange flows between 6 intranets, involving a total of 0.1 million hosts and 3 million flows. These flows are captured through the SPAN port of router, only containing summary information of every flow, but not including the specific contents of packages. Traces contain TCP, UDP and ICMP flows, being anonymized by mapping the IP addresses into network 10.X.X.X.

We let the worm break out at second 900. After a host has been infected, it sends an attack flow to a randomly chosen host every 30 seconds. A destination host will be infected if it is a vulnerable host, otherwise it won't. In the following experiments, we choose *0.1* as the fraction of vulnerable hosts in the network. Some information of experimental data is shown in Table 1. From Table 1 and Fig. 1 we can seen that, all the vulnerable hosts have been infected after 5000 seconds, and 41% of flows are attack flows.

Table 1. Three different worm scanning rates

Total flows (million)	5.08
Fraction of attack edges	0.41
Fraction of causal edges	0.0024
Fraction of vulnerable hosts	0.1
Fraction of infected hosts	0.100

When considering given parameter's effect to the algorithm performance, we only allow the corresponding parameter to change. Without special note, the initial values of the parameters in the experiment are shown in Table 2.

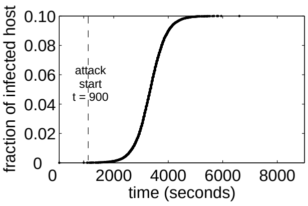

Fig. 1. Fraction of infected host along with time

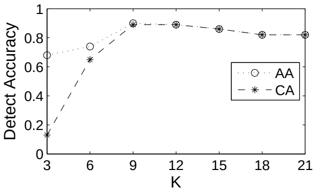

Fig. 2. *K* vs. *AA*, *CA*

Table 2. The initial values of parameters

Number of *accumulation process*: *K*	10
Time interval on the definition of precursor: *Δt* (seconds)	1000
Number of edges in the result set: *Z*	100
Running duration of online algorithm: *R* (seconds)	480
Size of sliding window: *S* (seconds)	2400

6.3 Preferences of Accumulation Algorithm

Parameter *K* Fig. 2 shows the impact of *K* on *AA* and *CA*. Algorithm performs the best when *K*=9, while *AA* and *CA* are both the highest. From this we can see that a good result only requires a few number of *accumulation process*. From Fig. 2 we also find that when *K* continuous increases, its accuracy declines a little. Because while accumulating along the reverse order of *causal chain*, excessive iterations of *accumulation process* makes the weight more likely to be aggregated to some normal edges before the worm breaks out, then more non-causal *malicious-destination edge* will enter *TOP-Z*.

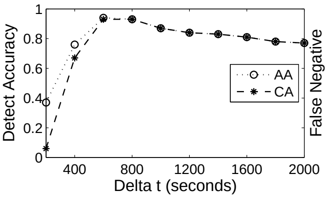

Fig. 3. *Δt* vs. *AA*, *CA*

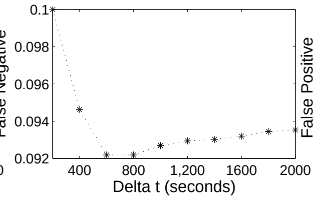

Fig. 4. *Δt* vs. *FN*

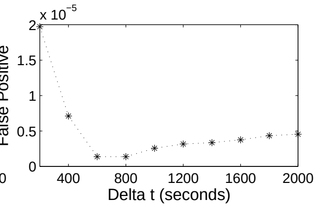

Fig. 5. *Δt* vs. *FP*

Parameter *Δt* Fig. 3 shows the impact of *Δt* on *AA* and *CA*. When *Δt* is increasing, detection accuracy climbs up but finally drops slightly. The accuracy is very low when *Δt* is very small, because reverse accumulation is more likely to arrive at a host that has no precursor in the previous *Δt* seconds, making some weights lost, reducing the possibility of pooling the weight into the top levels of *causal tree*.

Larger Δt makes the weight have more chance to accumulate to the top levels of *causal tree*, so detection accuracy will increase along with Δt. But *AA* and *CA* both lowered down when continuous increase Δt. This is because a wider Δt indicates more *precursors* – weight has higher possibility to be aggregated to some *normal edges* before the worm breaks out. We can also discover this from Fig. 4 and Fig. 5. The *FP* and *FN* are both increasing along with Δt, because of more *normal edges* are selected into *TOP-Z*.

From Fig. 2 and Fig. 3 we can see that, *CA* is usually very close to *AA*. This is because as the *accumulation process* proceeds, weights gradually accumulate to the initial *malicious-destination edges*. While at the initial phase of worm breaks out, there is not many infected host, the vast majority of *malicious-destination edge* are *causal edges*.

6.4 Preferences of Online Accumulation Algorithm

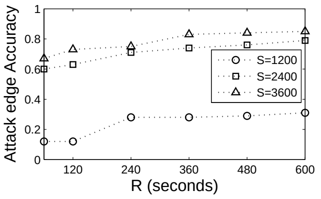

Fig. 6. R, S vs. *AA*

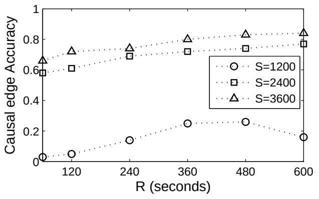

Fig. 7. R, S vs. *CA*

For an online algorithm, we hope its detect duration can be very short, so that worm propagation can be detected as soon as possible. Fig. 6 and Fig. 7 shows the impact of window size S and running duration R on detection accuracy. Using only partial data is not only an inevitable demand and benefits (reducing running time and memory consume) but also is a shortage (lower accuracy rate) for the online algorithm. As can be seen in Fig. 6 and Fig. 7, a bigger size of window leads to a higher detection accuracy, but there is only a little difference between the accuracy of S=2400 and S=3600. Increasing the running duration enables the algorithm accuracy to be increased. This is because when merging two trees form adjacent slide windows, more edges in the overlapping time interval will cause more conflict edges. Yet the change of accuracy with running duration is gentle, launching the *Accumulation Algorithm* every 60 seconds can achieve 70% accuracy.

6.5 Effect of Containment

Fast and accurate tracing worm source and initial propagation paths is essential to contain worms at the Internet scale. From Fig. 7 we can see that, when the parameters' value of online *Accumulation Algorithm* are R = 120, S = 2400, Z = 100, *CA* is at least

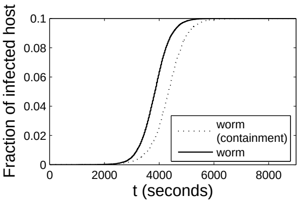

Fig. 8. Fraction of infected host at time *t*

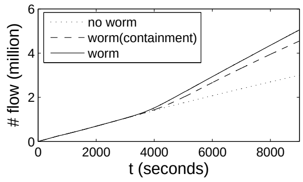

Fig. 9. Total number of flows before time *t*

60%. To illustrate the effects of worm containment, we conduct the following simulation experiments.

We add *h* infected hosts to the blacklist every 120 seconds (*h* = *minimal* {60, *total*×0.002}). Which '*total*' is the amount of infected hosts (excluding hosts already in the blacklist), '*minimal*{*x*, *y*}' return the smaller numerical value of *x* and *y*. Fig. 8 and Fig. 9 shows the simulation result. We can see thus containment delays worm spread about 500 seconds, while it reduces the network traffic by 10%.

7 Conclusions

Online tracing the evolution of a worm outbreak reconstructs not only patient zero (i.e., the initial victim), but also the infection node list in evolution process. Even if the proportion trails can be captured, it has significance in restraining evolution of worm in investigating and collecting evidence.

Tracing network worm propagation from the initial attack can inhibit continuous spread of the worm, ensuring that no more hosts is infected by the worm, and providing basis for the determination of worm attack origins. Experiment results indicate that the *Accumulation Algorithm* can achieve 90% detection accuracy.

References

1. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. IEEE Security and Privacy, Aug. 2003.
2. D. Moore, C. Shannon, and K. Claffy. Code-Red: A Case Study on the Spread and Victims of an Internet Worm. In Proceedings of Second ACM SIGCOMM Workshop Internet Measurement, page 273–284, 2002.
3. Moheeb Abu Rajab, Fabian Monrose and Andreas Terzis. Worm evolution tracking via timing analysis. In Proceedings of the 2005 ACM Workshop on Rapid Malcode (Fairfax, VA, USA, November 11, 2005). WORM '05. ACM Press, New York, NY, 52–59.
4. Pai Peng, Peng Ning, Douglas S. Reeves and Xinyuan Wang. Active Timing-Based Correlation of Perturbed Traffic Flows with Chaff Packets. ICDCS Workshops 2005:107–113.
5. Yinglian Xie, Vyas Sckar, David A.Maltz,Michael K. Reiter, and Hui Zhang. Worm Origin Identification Using Random Moonwalks. In Proceedings of IEEE Symposium on Security and Privacy, pages 242–256, May 2005.
6. A. Kumar, V. Paxson and N. Weaver, Exploiting Underlying Structure for Detailed Reconstruction of an Internet Scale Event. In Proceedings of ACM IMC, October 2005.

7. V. Yegneswaran, P. Barford and S. Jha. Global Intrusion Detection in the DOMINO Overlay System. In Proceedings of Network and Distributed System Security Symp. (NDSS), 2004.
8. M. P. Collins and M. K. Reiter, Hit-list worm detection and bot identification in large networks using protocol graphs. In Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007 (Lecture Notes in Computer Science 4637), pages 276–295, August 2007.
9. S. Sarat, and A. Terzis. On the detection and origin identification of mobile worms. In Proceedings of the 2007 ACM Workshop on Recurring Malcode (Alexandria, Virginia, USA, November 02, 2007). WORM '07. ACM, New York, NY, 54-60.
10. WAND Network Research Group. 2000. WAND WITS: NZIX-II trace data. http://wand.cs.waikato.ac.nz/wits/nzix/2/nzix-ii.php,July 2000.
11. Yinglian Xie, Vyas Sekar, Michael K. Reiter, and Hui Zhang. Forensic Analysis for Epidemic Attacks in Federated Networks. In Proceedings of the IEEE International Conference on Network Protocols, October 2006.
12. D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In Proceedings of 22nd Conf. Computer Comm., 2003.
13. M.E. Locasto, J. Parekh, A.D. Keromytis and S. Stolfo. Towards Collaborative Security and P2P Intrusion Detection. In Proceedings of Sixth Ann. IEEE SMC Information Assurance Workshop (IAW), page 333–339, June 2005.
14. N. Weaver, S. Staniford, and V. Paxson. Very Fast Containment of Scanning Worms. In Proceedings of Usenix Security Symp, page 29–44, 2004.
15. M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-End Containment of Internet Worms. In Proceedings of 20th ACM Symp. Operating Systems Principles (SOSP), Oct. 2005.
16. H.J. Wang, C. Guo, D.R. Simon and A. Zugenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In Proceedings of 2004 ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM), page 193–204, 2004.
17. Shad Stafford, Jun Li, and Toby Ehrenkranz, Enhancing SWORD to detect 0-day-worm-infected hosts, SIMULATION: Transactions of the Society for Modeling and Simulation International, vol. 83, no. 2, pp. 199-212, February 2007.