Lightweight Scheme for Generating Stealthy Probes

Shriram Ganesh, Adarshpal Sethi
University of Delaware
Newark, DE, USA
{ganesh, sethi}@cis.udel.edu

Rommie Hardy
US Army Research Laboratory
Adelphi, MD, USA
rhardy@arl.army.mil

Abstract— **Probing based approaches have been effectively used for network monitoring in the past. Probes such as ICMP pings provide an effective tool for detecting compromised nodes which try to delay or drop traffic. But an intelligent attacker may evade detection by giving preferential treatment to probe traffic. This is usually possible because probe packets have a different format from regular application packets and are easily distinguishable. The solution to this problem is to create stealthy probes which are indistinguishable from normal application traffic. In this paper, we build upon our earlier work on the design approaches for stealthy probing, and we propose a lightweight and effective scheme for generating stealthy probes.**

Keywords- stealthy probing; intrusion detection; network management; packet stamping.

I. INTRODUCTION

One promising approach for efficient and effective network monitoring is *probing* [1, 2, 3]. Probing based approaches involve sending test transactions over the network to monitor the health and performance of various network elements. Success or failure of these test-transactions depends on the success or failure of the network elements being tested. These test transactions are called *probes.* Probes are used to monitor a wide array of performance parameters including delay, loss, available bandwidth, traffic composition, routing behavior etc.

Probes such as ICMP pings provide an effective tool for detecting compromised nodes which try to delay or drop traffic. But such probes could be subject to differential treatment due to several reasons [13, 17]. A smart attacker who has captured a network node would delay or drop application packets to cause disruption in the functioning of a network, but the attacker would allow probe packets and their responses to pass through the compromised node in a normal manner. Using this technique, the smart attacker can avoid detection by the management system thereby rendering it ineffective.

In order to detect malicious attackers, there is a need to create stealthy probes which are indistinguishable from normal application traffic. When probes are made stealthy, the intruder is not able to differentiate the probes from regular application traffic. As a result, the compromised node ends up dropping or delaying both the probes and regular traffic, and the management system is then able to detect the presence of intrusion. In [17] we reviewed the design approaches for a mechanism to generate stealthy probes in a battlefield ad-hoc wireless environment. In this paper we build upon our earlier work and present a detailed scheme for generating stealthy probes. The scheme is robust to losses and delay which are characteristics of a battlefield ad-hoc network.

Stealthy probing allows a management system to be resilient to attacks by a malicious intruder. The primary goal of management is to ensure the health of a network including the detection of intrusions. If the management system itself falls prey to an intrusion, then it fails to fulfill its goals. Our scheme allows management traffic to flow without impediment even in the presence of intruders who have compromised some network nodes. Management survivability is crucial to network survivability and the use of stealthy probing is an important step in that direction.

The remainder of this paper is divided into five sections. We begin with an introduction to stealthy probing and cite some of the relevant work done in the area. We then describe the design decisions chosen from the alternatives that we had proposed in our earlier work. Next, we present the system design for our proposed stealthy probing scheme. We finally conclude with results from our simulations and outline possible future work in the area.

II. RELATED WORK

We presented a detailed survey of past approaches on stealthy probing in [13]. There are several limitations in the stealthy probing research done in the past. Many proposed approaches demand heavy instrumentation or added processing overhead at the intermediate nodes. The attackers may manifest themselves in innovative ways defeating many proposed strategies that provide defense only against specific attacks. Many proposed defenses become infeasible and ineffective due to the lack of deployment incentive at various areas of the network.

In the past, packet stamping has been used by many researchers for developing defense mechanisms against denial of service attacks. A survey of past work on packet stamping is presented in [17]. A packet stamping mechanism most relevant to our approach is proposed in [19], where the authors propose to use an access control mechanism, called Easy-pass, to prevent unauthorized access to network resources. A unique pass is attached to each legitimate IP packet. This pass is verified by an ISP edge router to provide access to the protected network resource.

We develop stealthy probes by stamping the IP packets in such a fashion that the probe traffic is indistinguishable from the regular traffic to all nodes not knowing the legitimate stamp

Prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

sequence. Like the proposed DDoS defense mechanisms, stealthy probing requires dynamically changing stamps, insertion of stamps at the source and verification at the destination. However, unlike the stamping mechanism used for the DDoS defense mechanisms mentioned above, routers en-route cannot be used to generate and/or validate the stamps. Also, unlike the DDoS defense mechanism, only the probe packets are stamped with the capabilities. Furthermore, the stamping needs to be stealthy to prevent unauthorized nodes from identifying stamped packets from unstamped packets. All these features help to make the scheme lightweight, an important consideration in battlefield ad-hoc networks.

III. DESIGN CHOICES

Various design issues need to be addressed while developing a stealthy probing mechanism in a battlefield ad-hoc wireless network. In [17] we discussed these issues and possible design choices in detail. Here we briefly state these design alternatives and the methods chosen for our algorithm.

- Probes can be sent separate from the regular application traffic or can be piggybacked on the regular application traffic. We choose to send probe packets separate from the regular traffic.
- In the two approaches described above for sending probes, care needs to be taken to ensure that the probes are still distinguishable at probing endpoints to identify and extract the probe information from the application packets. This probe identification could be done by using a pseudo-random sequence or by packet stamping. We choose the packet stamping mechanism in our algorithm.
- For the packet stamping mechanism, an essential property is to generate stamps which are hard to predict. The first approach is to have the sender and receiver agree on a key k and seed s. The initial stamp S is generated by using a function f() on k and s. Subsequent stamps could be generated by applying f() on the previous stamp value. The receiver uses the same mechanism to generate the stamps and uses a simple comparison operation to verify probe stamps. The second mechanism is to dynamically change packet stamps using a chain of one-way hash functions. We stick to the first approach in our algorithm.
- The process of updating the stamp at both sender and receiver can be done in several ways. Both sender and receiver can mutually agree on an update mechanism (time-based or traffic-based) and individually update the stamp value. Another approach to update stamp value is to make one node decide the next stamp value and inform the other node of the changed stamp. We use the mutual update mechanism in our algorithm.

IV. SYSTEM DESIGN

In this section we propose a design for implementation of stealthy probing. Probe traffic is designed such that the probe packets look similar to the application packets commonly present in the application traffic in the network under consideration. We disguise the probe packets to look like regular application packets (e.g. HTTP). Thus the probing application generates probe packets similar to the application packets and inserts a stamp in the IP header. We use a 16-bit stamp and place it in the IP identification field of the IP header. The receiver checks the received packets for valid stamps. On identifying a packet with a valid stamp, the packet is forwarded to the probing application. To avoid detection of any specific pattern in the stamps inserted in the probe packets the stamps are changed for each packet and regular application packets are also stamped with random values that do not form valid stamps.

A. Architecture

Figure 1 shows the architecture of our scheme with the network stack at both the probe generator (sender) and the probe receiver.

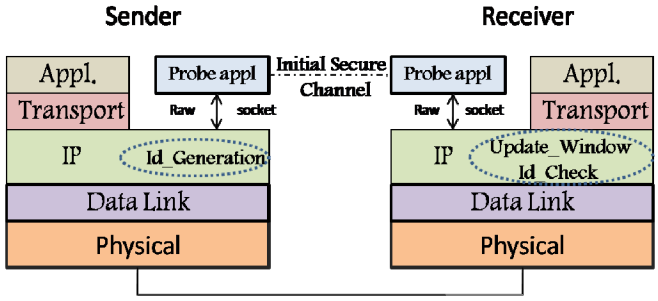

Figure 1. Network stack at Sender and Receiver.

B. Probe Sender

Algorithm 1 shows the procedure followed by the probe sender. The sender initiates the procedure by generating the key K, seed S for probe packets, seed S′ for application packets, and Δ, the average time to send 8 probe packets. The key and seed for probe packets are unique for each source-destination pair in the network. The sender communicates this information with the receiver making use of an initial secure channel. The seed for application packets is chosen arbitrarily by the node that sends these packets.

The seed S for probes is of 5 digits, starting from 0 and incremented by 1 for consecutive probes until 2^{15}-1 is reached, when it is wrapped around. The seed for application packets S' starts from 2^{15} and is incremented similar to probes until 2^{16}-1 when it is wrapped around back to 2^{15}. A 5 digit key value is interspersed into the seed and the resulting 10 digit value is passed as an input to the 32 bit version of the FNV1 hash function [18]. The resulting 32 bit hash value is XOR folded [18] to 16 bits so that the stamp can fit into the IP Identification field. The same procedure is used at the receiver end to create the window of acceptable stamps for the probe packets.

The probe application passes S and K along with the transport and application layer headers directly to the IP layer using a raw socket. The use of raw socket gives flexibility for the probe application to create contents of both the application and transport layer headers resembling any of the common

applications. The *Id_Generation* algorithm at the IP layer creates the ID field for every packet to be sent. We use the FNV-hash function [18] to create the ID field. FNV hashes are designed to be fast while maintaining a low collision rate. The high dispersion of the FNV hashes makes them well suited for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc.

Application packets on the other hand are passed to the IP layer from the transport layer using the regular socket interface. The IP layer generates the stamps for these packets using the same hashing algorithm that is used for probe packets, except that the seed S' is used instead of S.

Algorithm 1. Pseudocode for Probe Sender

Initialization:
Generate key for hashing K, seed S for probes, seed S' for application packets, and Δ, the average time to send 8 probe packets. Communicate K, S, and Δ to the receiver and receive acknowledgement making use of an initial secure channel. Send K, S, and S' to the IP layer using raw socket interface. The IP layer constructs the ID field for every packet to be transmitted.

Id_Generation:
1. **if** packet to be transmitted is a probe packet **then**
2. Set packet's ID field = FNV1_hash (K, S, 1)
3. $S = (S + 1) \bmod 2^{15}$
4. **else** // packet is an application packet
5. Set packet's ID field = FNV1_hash (K, S', 0)
6. $S' = (S' + 1) \bmod 2^{16}$
7. **if** $S' = 0$ **then** $S' = 2^{15}$ **end if**
8. **end if**

C. Probe Receiver

Algorithm 2 shows the pseudocode for the probe receiver. The receiver has the critical task of distinguishing the probe packets from the regular application packets. The receiver should also determine if the probe packets have a valid stamp. While validating the stamps, the receiver must take into consideration the losses and delay which are common in wireless ad-hoc battlefield networks. To address this issue, we maintain a *stamp_window* of 32 acceptable stamps. When the receiver node's IP layer receives a packet, the ID field of the packet is checked with the sequence of valid stamps stored in *stamp_window*. If there is a match, then the packet received is a probe packet. If the packet received was not a probe (because its ID field did not match any of the stamps stored in *stamp_window)*, then the packet is regarded as being an application packet and it is sent to the transport layer, similar to a normal operation of the IP layer.

When a received packet is determined to be a probe because its ID field matched a stamp stored in the *stamp_window,* it is also necessary to check if this is a new probe packet or a duplicate of a probe received earlier. This helps to guard against duplicate probe packets from an intruder who performs a replay attack on every packet received. Duplicate detection is performed by maintaining another window called *flag_window* of size 32. Each slot in the *flag_window* indicates the status of the corresponding stamp in the *stamp_window*, where a flag of 0 means that a probe with the corresponding stamp has not yet been received, while a flag of 1 indicates that the probe with that stamp has been received. Initially all slots in the *flag_window* are initialized to 0. Whenever a probe matches a slot in *stamp_window* the corresponding slot in *flag_window* is checked. If the slot in *flag_window* is 1, then this means that the probe received is a duplicate probe and it is dropped. Else, the probe received is a legitimate probe packet and the slot in *flag_window* is changed to 1. The valid probe contents are then sent to the probe application directly using a raw socket.

Each time a new probe packet is received, the *Update_Window* algorithm is invoked to update the *stamp_window* and *flag_window*. The next subsection explains the operation of the window scheme.

Algorithm 2. Pseudocode for Probe Receiver

Initialization:
When the values for K, S, and Δ are received from the sender, send acknowledgment to the sender. Send K, S, and Δ to the IP layer using raw socket interface. At the IP layer, do the following initializations:
1. **for** i = 0 **to** 31 **do**
2. *stamp_window* [i] = FNV1_hash (K, S, 1)
3. $S = (S + 1) \bmod 2^{15}$
4. *flag_window* [i] = 0
5. **end**
6. // Set tracker to point to current active octet of stamps
7. *tracker* = 0

Receiver_Loop:
This is executed for each packet received from the sender.
1. result = **Check_ID_Field** (recvd_packet)
2. **if** result = 0 **then**
3. // recvd_packet is a new probe packet
4. **Call Update_Window**
5. Set timer to expire at Δ
6. Pass packet using raw socket to probe application
7. **else if** result = 1 **then**
8. // recvd_packet is a duplicate probe
9. Drop recvd_packet
10. **else** // recvd_packet is an application packet
11. Pass packet to transport layer
12. **end if**

Algorithm 2 (contd). Pseudocode for Probe Receiver

```
Check_ID_Field (recvd_packet):
1.  for i = 0 to 31 do
2.      if recvd_packet.ID = stamp_window [i] then
3.          // this is a probe packet
4.          if flag_window [i] = 0 then
5.              // this is a new probe
6.              flag_window [i] = 1
7.              return 0
8.          else // this is a duplicate probe
9.              return 1
10.         end if
11.     end if
12. end
13. // reach here if no match with any stamp
14. // this is an application packet
15. return 2

Update_Window:
1.  if this is first probe packet then
2.      Set timer to expire at Δ
3.      return
4.  end if

5.  if timer is not expired then
6.      if all the first 8 stamps in window are received
7.          // (Case 1)
8.      or at least one stamp in the last 8 stamps of
            stamp_window is received // (Case 2)
9.      then
10.         if tracker <> 0 then
11.             tracker = tracker – 8
12.         end if
13.         Slide stamp_window and flag_window to
                left by 8
14.         Update stamp_window with next 8 stamps
15.         Insert 8 0's at end of flag_window
16.     end if
17. else // timer expired – Case 3
18.     tracker = tracker + 8
19.     if tracker > 15 then
20.         Slide stamp_window and flag_window to
                left by 8
21.         Update stamp_window with next 8 stamps
22.         Insert 8 0's at end of flag_window
23.         tracker = tracker – 8
24.     end if
25.     Set timer to expire at Δ
26. end if
```

D. Window Scheme

As explained in the previous subsection, the *stamp_window* and *flag_window* are used to determine whether or not a received packet is a valid probe packet. As successive probes are received and they match the successive stamps in the *stamp_*window, we must slide the windows to the left so the next sequence of stamps can be loaded from the right. To reduce overhead, we slide the windows in steps of 8 after receiving 8 valid probe packets and also load 8 new stamps on the right end of the window. This procedure would normally be sufficient if probes were not lost or delayed. In reality, probes are usually transmitted over an unreliable transport layer such as UDP. As a result, probe packets can get lost or delayed, and also probes can be received in a different order than the order in which the sender transmitted them. In the absence of any mechanism to deal with these situations, the window scheme can get confused or even get deadlocked. The purpose of the *Update_Window* algorithm is to help keep the window moving smoothly when such abnormal events occur.

When the first probe is received, a timer is started. This timer expires at Δ, which is the expected period of time to receive 8 probe packets. We maintain a *tracker* which points to a slot in the *flag_window*. The *tracker*, in conjunction with the timer mechanism, helps to account for the delay, loss, and clock synchronization issues in the network. The *tracker* tracks the currently active octet of stamps. Throughout the process we take care to see that the *tracker* does not move beyond the middle of the window. If it does, we slide the window to the left by an octet ensuring that the *tracker* is not beyond the middle in the new window that results. The sliding procedure comprises of two steps. In the first step we shift the stamps in *stamp_window* and the flags in *flag_window* to their left by 8 positions. In the second step we update the last 8 positions of *stamp_window* and *flag_window* with the next 8 stamps in the probe sequence and zeroes, respectively.

There are three conditions under which we slide the window. These conditions are described in the three cases that follow, and are also illustrated in Figures 2, 3, and 4 respectively. The figures show only the *flag_window* and the *tracker*, but it should be understood that the *stamp_window* is also updated in a corresponding manner. Similarly, the descriptions under each case below only mention the *flag_window* and the *tracker*, but again it is implied that the *stamp_window* is also updated in a corresponding manner.

Case 1: The first case is when all the stamps in the first octet are received before timer expiry. This means that these stamps have arrived without loss or significant delay. So we slide the window to the left by 8 slots. We also move the *tracker* to the left by 8 slots, so it still points to the same stamp that it pointed to previously. In Figure 2(a) the first eight probes have been received before timer expiry. Figure 2(b) shows *flag_window* after the sliding of the window and updating of the *tracker*.

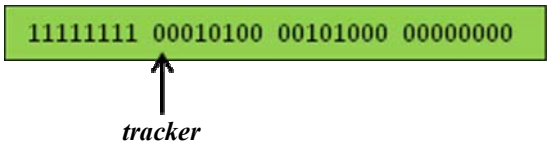

Figure 2(a). Case 1, t < Δ (timer not expired).

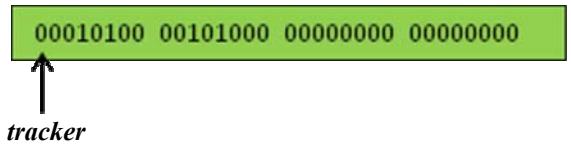

Figure 2(b). Case 1, after sliding of window.

Case 2: The second case is when at least one stamp in the last octet is received. This accounts for the case when some stamps in the first octet were lost or delayed in transmission so Case 1 did not occur. We slide the window as the missing stamps in the first octet are perhaps lost. In the unlikely event that they arrive at a later point of time, they will be dropped. This is the cost of keeping the window current so the active stamps from the sender can be recognized and accepted. In Figure 3(a) a stamp in the last octet is received before timer expiry. Figure 3(b) shows the final updated window. Once again, as in Case 1, the *tracker* is moved to the left by 8 slots, so it still points to the same stamp that it pointed to previously.

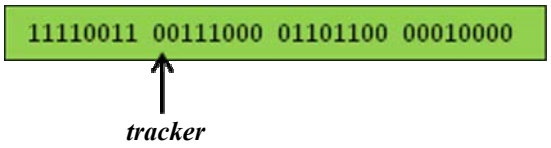

Figure 3(a). Case 2, t < Δ (timer not expired).

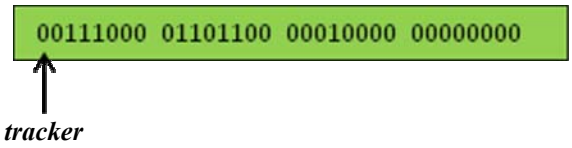

Figure 3(b). Case 2, after sliding of window.

Case 3: The third case is when the timer expires before either Case 1 or Case 2 occurs. In this case, the *tracker* is moved to the right by 8, but we do not slide the window. This allows extra time for some of the missing packets to arrive. However, if the *tracker* moves beyond the middle of the window, then the window is slid to the left by 8. This might happen if some of the initial stamps were lost or delayed significantly, in which case it is better to slide the window now without waiting for the old stamps. When the window is slid to the left by 8, the *tracker* is moved to the left by 8 as well, thus leaving it in the same position it was in before the timer expired. Figure 4(a) shows this situation. When the timer expires, the *tracker* will be moved to the right by 8. However, the *tracker* would then move beyond the middle of *flag_window*. So we slide the window and update *tracker* so it stays in the same position as when we started this case. Figure 4(b) shows the final window and *tracker* position.

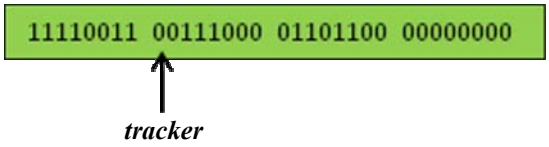

Figure 4(a). Case 3, t = Δ (timer expired).

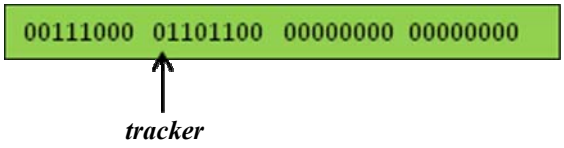

Figure 4(b). Case 3, after sliding of window.

V. SIMULATION RESULTS

We simulated the proposed scheme using OPNET Modeler 12.0 [20]. The network model consists of a source node connected to a destination node via an intermediate network modeled in OPNET as an Internet cloud. The source generates application packets and probe packets and injects them into the network. The network may lose packets, delay packets, or re-order packets on a random basis. The destination node for the probes implements the window scheme as discussed earlier. Below we discuss some of the results observed in this simulation.

The main performance measure monitored by us in the simulations was the percentage of probes lost due to the window scheme. We regard a probe as being lost by the window scheme when an arriving probe does not match any of the stamps in the *stamp_window*. The algorithm considers such an arriving packet to be an application packet. However, in the simulation, the source node marks all probe packets transmitted, and this mark can be checked at the destination. The mark is used to determine whether or not an arriving probe was incorrectly rejected (i.e. lost) by the window scheme because the ID field did not match the stored stamps. The failure to match happens when the packet was delayed and the *stamp_window* has moved on so it no longer contains the stamp of the packet being received at this time.

A. *Stamp Window Progress*

In Section 4.4, we described the three cases that result in a sliding of the stamp_window. The graph in Figure 5 shows which cases led to the sliding of the window at the receiver node as time progresses in the first 6 minute time slice of the 20 hour simulation. In this simulation experiment, 5% of the probes were delayed and 5% of the probes were lost in the network. A probe loss rate of 0.23% was observed due to the window scheme.

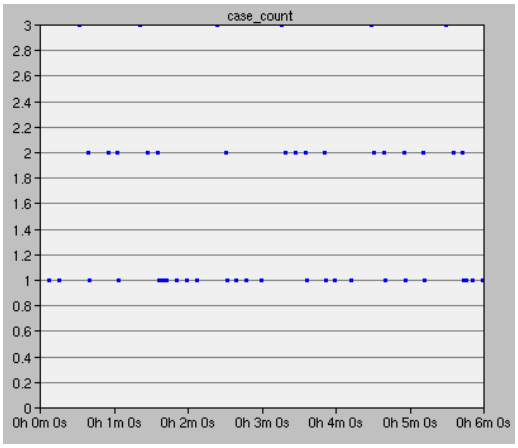

Figure 5. Sliding window progress.

B. *Variation in Delay and Simulation Duration*

We then analyzed the performance of the algorithm by varying the delay and simulation duration. The results are summarized in Figure 6. As the network delay increases, there is an increase in the loss of probes due to the window scheme but the loss does not go beyond 0.4% in the worst case even when 100% of probes are delayed. With increase in simulation duration the loss rate did not follow any pattern. The loss was 0 for the scenarios which had simulation duration of less than 10 hours and with less than 33% probes delayed. The results show that the scheme is robust to delay.

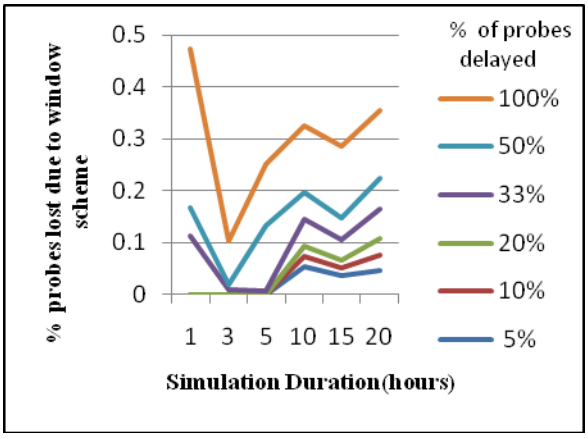

Figure 6. Variation in delay and simulation duration.

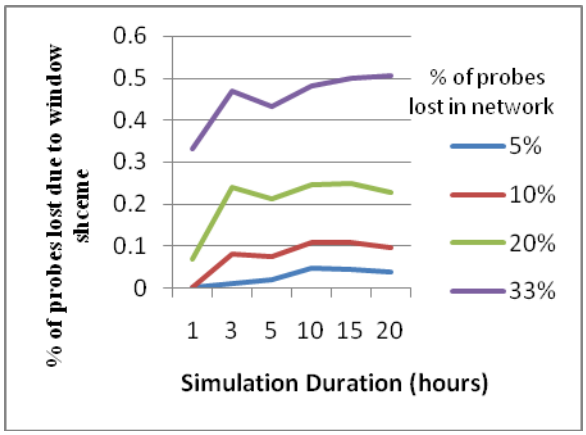

Figure 7. Variation in loss and simulation duration.

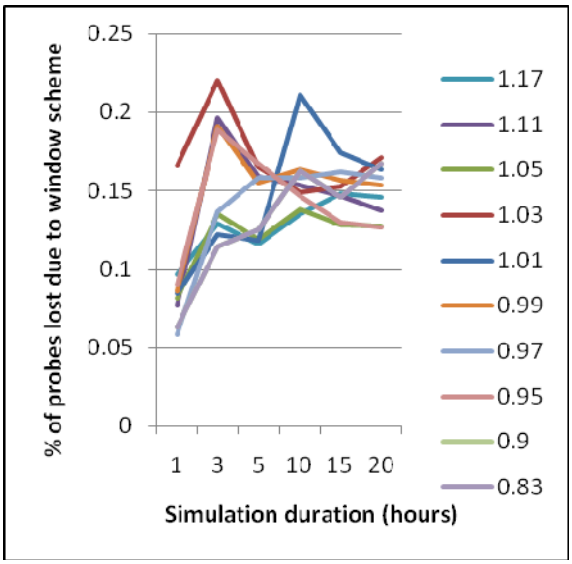

Figure 8. Variation in clock skew and simulation duration.

C. *Variation in Loss and Simulation Duration*

The next set of simulations was to evaluate the performance of the algorithm with variation in loss and simulation duration. The results are shown in Figure 7. As the loss rate in the network increases there is an increase in the probes lost due to the window scheme. The algorithm is robust for low loss rates. For example, below 33% loss rate in the network, the loss of probes due to window scheme is below 0.5%. When the loss rate in the network increases to 50% (not shown in this figure) the loss due to the window scheme becomes unacceptably high. Thus the results show that the scheme is robust when network loss rates are at reasonable levels (less than 50%).

D. *Variation in Clock Skew*

Since our scheme is based on an average rate of probing agreed between the probing endpoints, the variation in the

clock skew between the receiver and sender clocks should be taken into consideration. We ran simulations by increasing and decreasing the probing rate at the sender for practical variations in clock skew. The results are summarized in Figure 8. We find that our scheme is robust to variations in clock skew and the maximum percentage of probes lost due to the window scheme does not exceed 0.22%.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a detailed scheme for implementing stealthy probing. We build upon our earlier work on the design issues, and make choices from the alternatives proposed to build a mechanism to implement stealthy probing. The results show that the scheme is robust to loss, delay and variation in the clock skew. Our future work would be two-fold. First, we would like to address the assumption on the probing rate which is agreed between the probing endpoints. It should be possible to design a dynamic scheme in which the sender's rate of sending probes is not known to the receiver and may dynamically vary over time. Second, we would like to look into alternative schemes for implementing stealthy probing.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

REFERENCES

[1] M. Brodie, I. Rish, and S. Ma. Optimizing probe selection for fault localization. In DSOM-2001, IFIP/IEEE International Workshop on Distributed Systems Operations and Management, Nancy, France, Oct. 2001.

[2] M. Natu and A.S. Sethi. Active probing approach for fault localization in computer networks. In E2EMON'06, Vancouver, Canada, 2006.

[3] I. Rish, M. Brodie, S. Ma, N. Odintsova, A. Beygelzimer, G. Grabarnik, and K. Hernandez. Adaptive diagnosis in distributed systems. *IEEE Transactions on Neural Networks*, 6(5):1088–1109, Sep. 2005.

[4] A.B. Downey. Using pathchar to estimate Internet link characteristics. In ACM SIGCOMM, Cambridge, MA, 1999.

[5] J.C. Bolot. Characterizing end-to-end packet delay and loss in the Internet. *High Speed Networks*, 2(3), 1993.

[6] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet switched networks. *Performance Evaluation*, 27 and 28:297–318, 1996.

[7] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In ACM SIGCOMM 2002, Pittsburgh, PA, Aug. 2002.

[8] B. Huffaker, D. Plummer, D. Moore, and K. Claffy. Topology discovery by active probing. In Symposium on Applications and the Internet, Nara, Japan, Jan. 2002.

[9] M.J. Luckie, A.J. McGregor, and H.W. Braun, Towards improving packet probing techniques. In Internet Measurement Workshop, 2001.

[10] J. Postel. Internet Control Message Protocol. Request for Comment: 792, Sep. 1981.

[11] I. Avramopoulos and J. Rexford. Stealth Probing: Efficient Data-Plane Security for IP Routing. In Proc. USENIX Annual Technical Conference, Boston, MA, May 2006.

[12] V. Padmanabhan and D. Simon. Secure Traceroute to Detect Faulty or Malicious Routing. In Proc. ACM SIGCOMM HotNets Workshop, Oct. 2002.

[13] M. Natu, A.S. Sethi, R. Gopaul, and R. Hardy. Survey of Techniques for Robust and Secure Communication in Computer Networks. Technical Report No. 2007/337, Dept. of Computer & Information Sciences, University of Delaware, Newark, DE, Dec. 2006.

[14] A. Yaar, A. Perrig, and D. Song. StackPi: New Packet Marking and Filtering Mechanisms for DDoS and IP Spoofing Defense. *IEEE Journal on Selected Areas in Communications*, 24(10): 1853-1863, Oct. 2006.

[15] X. Yang, D. Wetherall, and T. Anderson. A DoS-Limiting Network Architecture. *ACM SIGCOMM Computer Communication Review*, 34(4): 241-252, 2005.

[16] A. Yaar, A. Perrig, and D.X. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In IEEE Symposium on Security and Privacy, 2004.

[17] S. Ganesh, M. Natu, A.S Sethi, R. Hardy and R. Gopaul. Design approaches for stealthy probing mechanisms in battlefield networks. In IEEE MILCOM 2008.

[18] http://isthe.com/chongo/tech/comp/fnv/

[19] H. Wang, A. Bose, M. El-Gendy, and K.G. Shin. IP Easy-pass: a light-weight network-edge resource access control. *IEEE/ACM Transactions on Networking*, 13 (6), Dec. 2005.

[20] OPNET Modeler. OPNET Technologies Inc. http://www.opnet.com