DESIGN APPROACHES FOR STEALTHY PROBING MECHANISMS IN BATTLEFIELD NETWORKS

Shriram Ganesh, Maitreya Natu, Adarshpal Sethi
University of Delaware
{ganesh,natu,sethi}@cis.udel.edu

Richard Gopaul, Rommie Hardy
US Army Research Laboratory
{rgopaul,rhardy}@arl.army.mil

ABSTRACT*

Various approaches have been proposed in the past for monitoring a network to diagnose failures and performance bottlenecks. One such approach for efficient and effective monitoring is probing. Probes such as ICMP pings are an effective tool for detecting network nodes which have been compromised by an attacker who tries to delay or drop traffic passing through the captured node. However 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. Hence, it is important to probe in a stealthy manner so as to avoid identification of probes by an attacker and to ensure the collection of accurate system health statistics. In this paper, we review design approaches for generating stealthy probes and describe various possible mechanisms that can be used for such a design. These approaches are evaluated according to the design criteria and we identify what may be feasible solutions for stealthy probing in battlefield ad-hoc wireless networks.

1. INTRODUCTION

With the objective of providing seamless end-to-end services, various approaches have been proposed in the past for monitoring a network to diagnose failures and performance bottlenecks. One promising approach for efficient and effective 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, called *probes*, depends on the success or failure of the network elements being tested. Probes can be of various types such as one-packet probes [4], packet-pair probes [5, 6], packet-train probes [7] etc. Probes are used to monitor a wide array of performance parameters including delay, loss, available bandwidth, traffic composition, routing behavior etc. Currently, *pings* and *traceroutes* are the most popular probing tools to detect network availability. Examples of various other probe uses include Jacobson's [4] one-packet probes in *pathchar* to estimate link bandwidth from round trip delays of different sized packets sent to successive routers along a path. *Skitter* [8] uses *traceroute* like probes for tracking Internet topology. *PathChirp* [9] uses packet-pair probes for estimating available bandwidth on a path.

* 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.

Most probing methods employ probes that do not resemble non-probe traffic generated by regular applications. Such probes might experience different network conditions as compared to regular application traffic. There could be various reasons for the different treatment of probe traffic. Probes that require intermediate nodes to do processing (e.g. *traceroutes*) might be given lower priority by the routers. End-hosts might block certain protocols (IPMP [10], ICMP [11]) used by probe traffic.

Another more daunting possibility is the identification of probe traffic by a malicious entity within the network that creates the illusion of a healthy network service for the probe traffic. Consider an attacker that probabilistically drops or delays packets of regular application traffic. Such an attacker can manage to stay undetected by giving preferential treatment to probes to keep the probe stations unaware of the malicious drops and delays along the probe path. Another example could be of malicious routing [12]. Probes can be used to detect the inconsistency between the advertised and actual routes for end-to-end paths affected by an attack. However if an attacker can identify the probe traffic, it can treat probes in a different manner, possibly allowing probes to pass through the correct routes, thus avoiding detection by the probes.

978-1-4244-2677-5/08/$25.00 ©2008 IEEE

Hence, there is a need for methods to conduct probing in a stealthy manner preventing the intermediate malicious nodes from misleading the probing nodes by treating the probe packets differently from normal packets. We refer to this probing approach as *Stealthy Probing*. When probes are made stealthy, the intruders are not able to distinguish probe traffic from regular application traffic. So they drop both probe as well as application traffic making it possible for the probing nodes to detect the presence of intrusion.

In this paper, we identify design issues for a mechanism to generate stealthy probes. We also propose alternative design strategies for probes that are similar to regular application traffic but are constructed in such a manner that only the sender and recipient of the probes can distinguish the probe traffic from the application traffic. We also will discuss various issues involved in constructing the stealthy probes for a wireless network environment and present possible design approaches for implementing such probes. The main contribution of this paper is that it advances the state of the art in stealthy probing designs by proposing various approaches to the solution of the problem. These approaches are evaluated according to the design criteria and we identify what may be feasible solutions for stealthy probing in battlefield ad-hoc wireless networks.

2. RELATED WORK

In the past, very little work has been done on stealthy probing. Avramopoulos and Rexford in [13] propose a light-weight data plane approach by creating an encrypted tunnel between two routers and diverting both probe and regular traffic into the tunnel to make probe and regular traffic indistinguishable. Secure traceroute proposed by Padmanabhan and Simon [14] securely traces the path of the existing traffic, thus preventing the routers from misleading specialized *traceroute* packets by treating them differently from normal traffic packets. We present a detailed survey of past approaches on stealthy probing in [15].

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. The growing use of wireless networks opens another array of vulnerabilities to address while developing stealthy probing strategies.

In the past, packet stamping has been used by many researchers for developing defense mechanisms against denial of service attacks. Yaar *et al* [16] propose StackPi, where a 16-bit path identifier (Pi) is assigned to each packet and is stored in the IP identification field. The Pi-marks are generated as the packet flows along its path to its destination. All packets traversing the same path receive the same marking. Yang *et al* [17] propose a mechanism where the sender first acquires a token from the destination, representing the capability to send, and then stamps the packets with the capability. These packets are verified at the intermediate verification points on the way to the destination for the presence of the valid capability. A capability is valid for a limited time and packet count, and needs to be renewed for further communication. Capabilities are generated by the destination using a chain of one-way hash functions. Careful design and evaluation of a more complete capability-based architecture, called Traffic Validation Architecture (TVA), is proposed by the same authors in [17]. Another capability-based mechanism is proposed in [18], where routers provide path specific information that is aggregated by the destination to generate a capability. The sender inserts this capability in the subsequent packets. Routers on the way verify the capability for the correctness of their part of the information in the aggregated capability. Routers change their capability marking periodically and the new capability value is communicated to the sender. Routers keep a window of keys as valid at any one time. The packet stamping approach 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 propose to develop stealthy probes by stamping the packets in such a fashion that the probe traffic is indistinguishable from the regular traffic to all nodes not knowing the legitimate stamp 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 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.

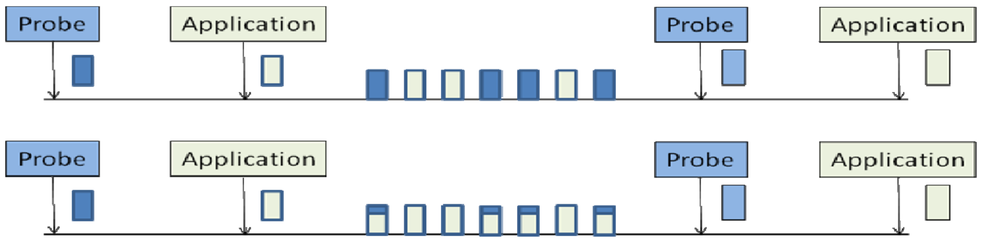

Figure 1 (a) Probe and application packets sent as separate packets
(b) Probes piggybacked on application packets

3. DESIGN ISSUES

Various design issues need to be addressed while developing a stealthy probing mechanism in a battlefield ad-hoc wireless network. Below we describe these issues and explain the rationale behind our choice of system design.

3.1 How to send probes

Probes can be sent separate from the regular traffic or can be piggybacked on the regular application traffic. There are various design issues involved in both approaches to keep the probes stealthy.

As shown in Figure 1a, one design approach is to develop probes resembling application traffic on the desired probe path and insert the probe packets within the stream of application packets giving the impression that probe packets are actual application traffic. Thus probe packets need to be designed to have the same protocol and port numbers, and consistent sequence and acknowledgement numbers as the rest of the application traffic. A challenging task is to maintain consistent sequence and acknowledgement numbers between the probe and application packets as well as filtering and extracting probe traffic from the real application traffic.

Another approach to insert probe traffic stealthily could be to piggyback probe traffic over the already existing application traffic as shown in Figure 1b. In this approach, instead of creating new packets, probe information is embedded into the application packets that are already being sent over the probe path. This approach avoids the problem of developing new probe packets and making them consistent with the regular application traffic. However, care needs to be taken while embedding the probe packets on the application packets to ensure that the application packets (header and data) stay consistent with the regular application traffic. There is also a need for some technique to extract the probe information from application packets at the destination. A drawback of this approach is that it depends on the availability of existing application traffic to probe a certain network path. Thus there still is a need for developing separate probe packets resembling application traffic to be sent on a probe path when regular application traffic is infrequent.

3.2 How to distinguish probe traffic from regular traffic

In the two approaches described above to solve the problem of making probes indistinguishable from regular application traffic, care needs to be taken to ensure that the probes are still distinguishable at probe endpoints, otherwise the destination will not be able to identify and extract the probe information from the stream of application packets. If the probes are sent as separate packets, a mechanism is needed at the destination node to identify the probe packets from the application packets and likewise, when a response is generated, a similar mechanism is needed for the source to identify the probe responses. Similarly, if the probes/responses are piggybacked over the application packets, the destination node must be able to identify the application packets that contain piggybacked probe information. Once such packets are identified, the destination node needs a way to extract the piggybacked probe information from the application packets. It is also important that this ability to distinguish probes from application packets be limited to the source and destination nodes. Intermediate nodes must not be able to make this distinction; otherwise the probes will not be stealthy. Also, since various sources may need to probe various destinations, any of the network nodes may be potential probe end-points. For this reason, the scheme cannot rely on secrecy of design to provide stealthiness. In the following sub-sections, we discuss various ways to identify stealthy probes at the probing end-points.

3.2.1 Using pseudo-random sequence

One possible approach could be to use a pseudo-random sequence to insert probe information in the application traffic. The two end-nodes pre-agree on the seed *s* to generate the pseudo-random sequence. The two end-nodes then generate the pseudo-random sequence starting with the seed *s* and insert the probe information into the appropriate application packets based upon this sequence as shown in Figure 2. Figure 2a shows a naïve scheme in which probes are inserted in regularly spaced packets, while Figure 2b shows how the pseudo-random sequence is used to select the packets that contain probes. Probes are normally piggybacked on the application packets. In case of unavailability of application traffic, independent probe packets are sent making the packets resemble application traffic. Each probe packet (piggybacked or independent) contains information about the location of probe information at a predefined location in the packet. After identification of the probe packet based on the pseudo-random sequence, the destination node uses this information to locate the probe information within the packet. This approach is simple and easy to implement; however it is vulnerable to packet losses and reordering. If some packets in the sequence are lost or are delayed so they reach the destination after packets transmitted later, the destination will not be able to identify the probes correctly. For this reason, this scheme should only be used for applications that use a reliable transport mechanism such as TCP.

3.2.2 Using packet stamping

Another implementation approach is to stamp the probe packets with a ticket that is known to both the source and destination nodes. This approach, unlike the previous approach using pseudo-random sequence, is robust to loss, delay, and reordering of packets. As in the previous approach, in the presence of application traffic the probes are piggybacked over the application packets. In the absence of application traffic, separate probe packets are sent making the packets resemble the regular application packets. In both cases, the packet that contains a probe is stamped by the currently valid stamp. The packet contains the probe information at a predefined place in the packet. This information is used by the destination node to extract the probe out of the received packet.

The packet stamps for this purpose must have the following properties:

- Inconspicuous: Stamping should be inconspicuous to prevent detection of stamps and consequently the probes by the attacker. Stamps should thus not require use of additional fields in the network packets that are not used by regular application packets. One approach to meet this requirement is to use a header field of some layer in the existing network stack, e.g., the identification field, or type of service field in the IP header. Furthermore, packets of the regular network traffic should also be stamped with random values to avoid detection.
- Dynamically changing: Using the same stamp value for all probe packets reveals a traffic pattern that can be used by the attacker to detect probe traffic. Hence stamps should be changed dynamically.
- Hard to predict: Stamps should be changed in such a manner that the attacker cannot infer any pattern in the changed stamps. Identification of any such pattern can allow the attacker to predict the next stamp and thus detect the next probe packet.
- Lightweight: The stamp generation, embedding, and verification process should not incur significant computational overhead or network traffic.
- Robustness: Stamping verification should be robust to loss and reordering.

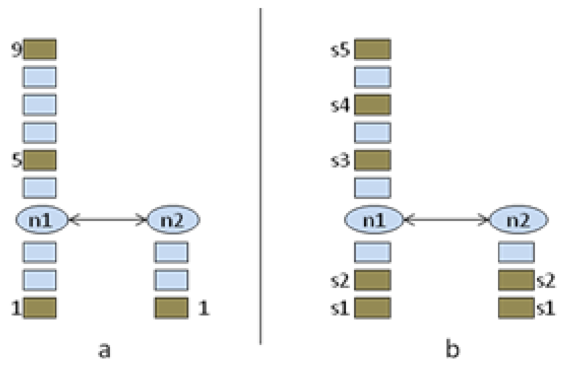

Figure 2: (a) Probes inserted at positions 1,5,9,.... (b) Probes inserted at random locations using stamps s1, s2, s3, s4, ...

3.2.3 Generation of a series of stamps

In this section, we present approaches to dynamically change the stamp values, while preserving the proposed stamp properties. The objective here is to be able to change the stamp dynamically, at the same time making both communication ends aware of the changed stamp. While changing the stamp, it is important to make the next stamp value hard to predict to prevent an attacker from identifying the stamp and thus identifying the probes. In order to change the stamp value periodically we propose two approaches.

In the first approach presented in Figure 3a, the source and destination node agree upon a seed *s* and a key *k*. Starting with the first stamp *s*, both ends build a series of stamp

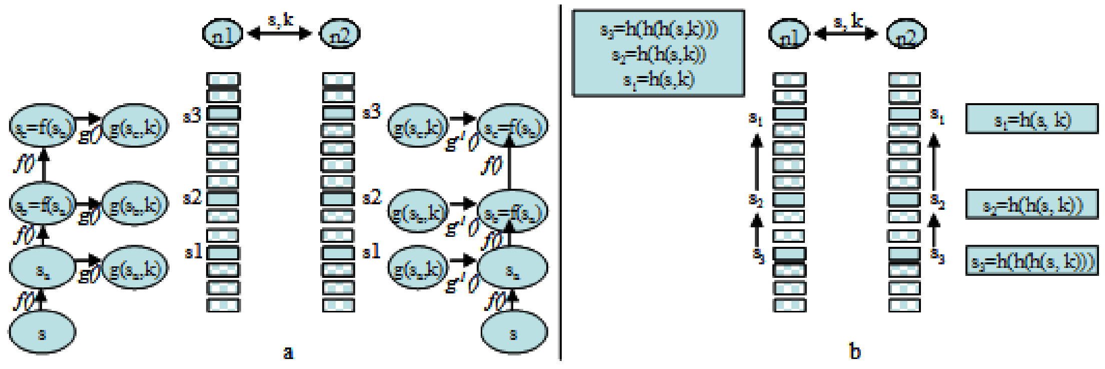

Figure 3: (a) Sequence of stamps generated using function f() and encrypted using function g(). (b) Sequence of stamps generated by performing multiple iterations of one-way hash function h() and using the stamps in the reverse order from the sequence.

values by applying a function *f()* on the previous stamp value. The sender encrypts its stamp value using a function *g()* and the key *k*. The receiver decrypts the stamps by applying the inverse hash function $g^{-1}()$ with the key *k*. The receiver then verifies the decrypted stamp value with the stamp series computed at its end using the function *f()*.

We present another approach shown in Figure 3b which is based on dynamically changing packet stamps using a chain of one-way hash functions. As in the previous approach, the two end-nodes agree on the initial seed *s* and the hash key *k*. The client then builds a chain of values by repeatedly applying the hash function on the previous value. Thus starting with a seed *s*, the client builds a series of values *h(s,k), h(h(s,k)), h(h(h(s,k))),* and so on. The sender then uses these stamps in the reverse order in this sequence, thus using the stamp *h(h(h(s,k)))* before *h(h(s,k))* before *h(s,k)*. Once the list is exhausted, the sender restarts with the last element of the list as the next stamp. The destination stores the initial seed *s* and the hash-key *k*. The destination performs multiple hash operations on the seed *s* to obtain the currently active stamp and compares the incoming packets with the desired stamp value to identify probes.

3.2.4 Changing stamp value

For implementation of a dynamically changing stamping mechanism, the life-time of a stamp could be time-based [18] or traffic-based [17]. That is, the stamps could be changed after expiration of a certain time interval or after exchange of certain number of packets. In the past, researchers have used both approaches while proposing dynamically changing stamps. Time-based stamps avoid the overhead of maintaining packet counts required for traffic-based stamps. However, as time-based stamps allow a client to use the stamp to send as many packets as desired within the stamp life-time, this approach could reveal a traffic pattern if large amounts of probes are sent within the stamp life-time.

The process of updating the stamp at both sender and receiver can be done in several ways. (1) Both sender and receiver can mutually agree on an update mechanism (time-based or traffic-based) and individually update the stamp value. To update the stamps after a certain time interval, the sender and receiver need to have synchronized clocks. For traffic-based stamp update, sender and receiver need to maintain traffic statistics and also address the scenario of dropped and delayed packets. (2) 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. For instance, the receiver node can inform the sender of the new stamp after expiration of certain time interval and/or after receiving certain number of probe packets. This mechanism however involves additional communication overhead between the two end-points.

4. PROPOSED SYSTEM DESIGN

In this section we propose a design for implementation of stealthy probing. We propose to generate probe traffic that looks similar to the application traffic. However, we stamp the probe packets with stamps known only to the sender and receiver. Probe traffic is identified at the receiver by identifying the valid stamps. The generation and insertion of stamps is done in such a manner that attackers cannot identify probe packets to be any different from regular application packets. We first discuss the procedure of sending and receiving probes. We then present our approach to generate, embed, and verify stamps.

4.1 Generation of probe traffic

Probe traffic should be designed such that the probe packets look similar to the application packets commonly present in the application traffic in the network under consideration. In the scheme proposed here, we consider HTTP traffic to be present as regular application traffic in the network and hence disguise the probe packets to look like HTTP packets. Thus the probing application program generates probe packets similar to HTTP 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 IP identification field is used by the routers to identify fragments of a packet during fragmentation. This is not a problem because even if the packet is fragmented en-route, the fragments would be reassembled by the IP layer at the destination before the stealthy probing algorithm uses the ID field to detect a received probe packet.

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. Figure 4 presents the steps involved in sending and receiving the probe packets. This figure shows a probing application sending and receiving probes using UDP port 1600. The sender disguises probe packets as HTTP packets, but inserts a stamp in the IP identification field. The receiver on observing a valid stamp extracts the probe part of the packet and forwards it to the appropriate probe receiving application.

4.2 Stamping

We now present our approach to generate stamps. The stamps are designed to have the following properties:

- Stamps are 16 bits long so that the stamp can be inserted in the IP identification field.
- Stamps are made unique for each packet to avoid detection of any pattern in the probe traffic.
- The change in the stamps is made difficult to predict.

We present two approaches of generating the stamp sequence.

4.2.1 Approach 1: Sender and receiver independently generate stamp sequence.

The sender and receiver exchange a secret *k* and an initial seed *s*. As shown in Figure 3a, both sides then generate a sequence of stamps using a function *f()*. This function builds the new stamp from the previous stamp value. The resulting stamp value is then encrypted using the secret *k*. Each outgoing probe packet is stamped with a unique stamp value encrypted in this manner.

Both the sender and receiver maintain a window of *w* stamps. These *w* stamps represent currently active stamps. The sender uses the window to ensure that a random value inserted as stamp in the regular application packets is not present as a stamp in the window. This check prevents a regular application packet from being mistaken to be a probe packet at the receiver. The receiver maintains a window to accept out of order probe packets.

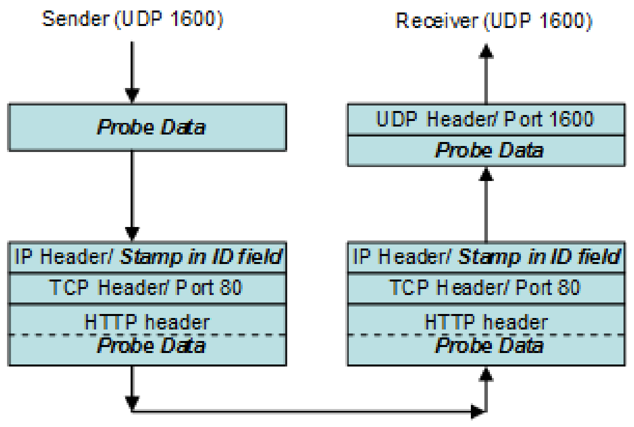

Figure 4: Steps performed to generate and receive probe traffic similar to http traffic

4.2.1 Approach 2: Only receiver generates stamp sequence

In the second approach stamps are generated only at the receiver. The receiver generates a sequence of stamps by performing multiple iterations of a one-way hash function and then using the last stamp in the sequence first. The sender performs an initial handshake with the receiver to obtain the first stamp and a secret *k*. One stamp is used for multiple packets by the sender. However, to insert a different value in the ID field for each probe packet, the sender hashes the stamp with the secret k and some of the packet content that changes with every packet but does not change for a single packet from source to destination. One example of such a field is the TCP sequence number. The receiver performs a similar operation on the received packet to verify the stamp. The receiver updates the stamp value after expiration of *t* seconds and updates the sender with the new stamp. To accept out of order probe packets, the receiver accepts packets with the current stamp or a previous stamp value.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed various design approaches for a mechanism to generate stealthy probes. We have also discussed possible mechanisms for identification of probes at probing endpoints. Finally we have proposed a system design and discuss how the earlier approaches could be applied to such a design. Our future work would be to look at other possible system designs and to implement and test their correctness and effectiveness in a wireless ad-hoc battlefield environment.

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] V.J. Ribeiro, R.H. Riedi, R.G. Baraniuk, J. Navratil, and L. Cottrell. PathChirp: Efficient available bandwidth estimation for network paths. In Passive and Active Measurement Workshop, 2003.

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

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

[12] P. Kruus, D. Sterne, R. Gopaul, M. Heyman, B. Rivera, P. Budulas, B. Luu, T. Johnson, and N. Ivanic. In Band Wormholes and Countermeasures in OLSR Networks. In SecureComm2006, Baltimore, MD, Aug. 2006.

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

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

[15] 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.

[16] 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.

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

[18] 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.

[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.