

# ANET: An Anonymous Networking Protocol

Casey Marshall  
csm@soe.ucsc.edu

May 31, 2005

## Abstract

This paper presents a simple, anonymizing network protocol. Its primary goal is to provide untraceability of connections between two parties, by routing messages securely through a loosely organized, untrusted peer-to-peer network. This paper is primarily a companion to the reference implementation of this protocol, as freely redistributable Java source, and is intended to fulfill the project requirements for CMPS 223: Advanced Computer Security, at the University of California, Santa Cruz.

## 1 Introduction and Related Work

ANET is a simple, anonymizing, general-purpose network protocol. It uses a distributed, ad-hoc peer-to-peer network to route messages between peers. The protocol is message-based, and does not guarantee that messages will arrive in the order they are sent, if at all — in which regard it is similar in intended usage to UDP. As such, it only requires the most basic underlying transmission method, such as UDP or TCP (TCP can be used, even though it does not preserve message boundaries, because the protocol itself will preserve those boundaries). The class of anonymity we aim for here is *sender-recipient* anonymity, where the mere fact that two peers are in communication is hidden from the rest of the group, and from outside observers.

ANET is not intended to be a bulk data transfer protocol, as it is far too inefficient. It is, instead, meant to preserve as much anonymity as possible in the face of collaborating peers.

Mixes [2] is perhaps the oldest notion of anonymous communication using public-key cryptography, and a number of modern protocols can be thought of as descendants of this. Onion routing [5] is an example: it gets its name from the layers of encryption applied to messages when they are routed, and how a layer is removed on every hop. These protocols aim to establish *sender anonymity*, where the server has no information about who is making the request.

Crowds [4] is possibly the protocol that most resembles the present one; crowds relies on the same assumptions as ANET for its security — that two peers can create a private channel between them by becoming lost in the noise

of a crowd: here, a network of peers. Crowds is interesting because it has amongst its design goals reasonable performance for web transactions.

Six/Four [3] is another anonymizing, general-purpose network protocol, whose primary goal is to use such a network to provide citizens of states (China being the typical example) access to information that the local censorship policy would normally prevent access to. Notable features of six/four is its attempts at masking the existence of the network, by making nodes appear to be web servers using the HTTPS protocol; the primary disadvantage of six/four is its reliance on “trusted peers,” which make the system much more vulnerable to attack and censorship than it could otherwise be.

## 2 Informal Description of the Protocol

Each peer is identified by a triplet in the system: a public RSA key, an internet address (in this version, an IPv4 address and a port number), and a 16-byte identifier. Each message is encoded in a *packet*, which has the following structure:

```

uint16    time-to-live
byte[16]  last-hop
byte[16]  nonce
uint16    payload-length (n)
byte[n]   payload

```

The general algorithm each peer uses when receiving a packet is:

```

ANETRECEIVE()
1  while true
2      do (packet, source-address) ← READDATAGRAM()
3          if packet.ttl > 0
4              then packet.ttl ← packet.ttl - 1
5                  next-hop = {PEERS ∩ {packet.last-hop}
                              ∩ {source-address}}R
6                  packet.last-hop ← source-address
7                  SENDDATAGRAM(next-hop, packet)
8                  MAYBECRYPT(packet.payload)
9                  MAYBEASYMMETRICDECRYPT(packet.payload)

```

That is, whenever a packet is received, it is immediately forwarded to another random peer if the *time-to-live* field is nonzero; the packet will never be forwarded to either the previous peer, or the next-to-last peer. After the packet is forwarded, if its connection ID is that of the currently active connection, the payload is processed with the symmetric encryption and MAC key of the session; otherwise, an attempt is made to decrypt the payload with the private

RSA key. If the packet is a *connection-request* message, then the payload will have the following form, encrypted with the destination peer’s public key:

$$\{N, A, B, pre-master-secret\}_{PK_B} \quad (1)$$

$N$  is the nonce in the enclosing packet, and  $A$  and  $B$  are the identifiers of the connector and connectee. Nothing else but the presence of a payload encrypted with the receiving peer’s public key signifies a connection request.

This payload establishes a connection between *source-peer* and *dest-peer*. Additionally, the payload for such messages will be padded with a random amount of random data (up to the limit of a packet size), to foil observations on the length of the packet in determining a connection request. Since the ciphertext in RSA is always based on the size of the key, the boundary between encrypted message and padding is implicit. If *dest-peer* accepts the connection, it generates four keys like so:

$$\begin{aligned} K_0 &\leftarrow \text{SHA-256}(pre-master-secret) \\ K_1 &\leftarrow \text{SHA-256}(K_0) \\ K_2 &\leftarrow \text{SHA-256}(K_1) \\ K_3 &\leftarrow \text{SHA-256}(K_2) \end{aligned}$$

And sends a reply message, encrypted with the public key of the source peer ( $N$  is again the nonce from the enclosing packet):

$$\{N, A, B\}_{PK_A} \quad (2)$$

Otherwise, the payload is encrypted and authenticated with two of these four keys (the first two keys comprise the connecting peer’s encryption and MAC keys; the last two the accepting peer’s encryption and MAC keys). The payload will have the following format:

$$\begin{aligned} C &= \text{AES-CTR}(nonce, M, K_E) \\ payload &= \{C, \text{HMAC-SHA-256}(C, K_M)\} \end{aligned}$$

Sending a packet to a peer, given the *payload* as constructed above, is a straightforward process:

ANETSEND(*payload*)

- 1 *packet.payload*  $\leftarrow$  *payload*
- 2 *packet.ttl*  $\leftarrow$   $\{512 \dots 1024\}_{\mathcal{R}}$
- 3 *last-hop*  $\leftarrow$   $\{\text{PEERS}\}_{\mathcal{R}}$
- 4 *next-hop*  $\leftarrow$   $\{\text{PEERS} \cap \{\textit{last-hop}\}\}_{\mathcal{R}}$
- 5 SENDDATAGRAM(*packet*)

### 3 Security Analysis

The goal of this protocol is to allow two peers to establish a secure connection with one another, without exposing to any of the other peers nor an outside observer the fact that these two peers are communicating with one another. We believe that we are on solid cryptographic ground here, except in proving the identity of the connecting peer. A further refinement to the protocol would be to introduce a digital signature created by the connecting peer in the connection attempt.

However, our more immediate goal is to guarantee that the presence of a connection between two peers cannot be determined by other observers, within the network or without. Specific features of this protocol that try to provide this are:

- The random initial time-to-live. This is intended to make tracing the path of a packet back to its true source infeasible. Given that each path is between 512 and 1024 hops, and  $n$  peers in the network, the number of possible paths is

$$\sum_{i=512}^{1024} (n-3)^i$$

We subtract three because a peer never forwards a packet to itself, to the peer that immediately sent the packet, or to the peer two hops ago. This is not a computationally complex number, but still guarantees that it is not possible to say, within a reasonable degree, that two nodes were in communication.

- The policy of always forwarding messages to a new random peer, regardless of whether or not it has reached its destination already. This should help foil observational attacks where a packet is seen to be sent to a peer, but not forwarded because it was the final destination.
- The lack of identifying values sent in the clear.

One thing we are sure we cannot accomplish with this (or indeed, perhaps *any*) protocol is perfect anonymity if a certain percentage of the peers in the network are “corrupt” — that is, if a certain number of peers are collaborating to trace the communication of messages, they can trace messages between two peers with certainty. Reiter and Rubin are aware of this [4], and make the argument that within a certain limit, their protocol still achieves an adequate level of anonymity.

Wright et al argue [6] that outside of these assumptions, attacks exist that can still undermine the anonymity of a number of protocols, typically with approximately polynomial complexity. What the present protocol aims to provide, through its complete rejection of path re-use, and using a completely random path for every message, is resistance to such analysis.

As Reiter and Rubin point out, a *local eavesdropper* — that is, someone able to monitor all communications going into and out of a peer — will be able to tell, with certainty, when a peer initiates a connection. ANET too is vulnerable to such observations, because even though most attributes of the packet sent are random, it is simple to observe that no packet with those attributes was received by a peer, but one was emitted. This attack could be foiled as well, if peers periodically, but at random intervals, send out “dummy” packets whose only purpose is to bounce around until they expire.

### 3.1 Trust

We have not defined any notion of trust in this protocol. Our presumption in this initial version is that trust is established among the peers before they begin using the protocol — that is, that each peer has the public keys of all the other peers, and that each peer has assigned a trust label to peers that he has confirmed the trust of. Thus, we follow roughly the PGP trust model, as peers must establish trust through different channels.

## 4 Reliability

One possible cause for concern in this protocol is the definite possibility of messages being dropped as their time-to-live drops to zero. Aside from performance impact, this should not preclude the general use of this protocol.

Given the rather large time-to-live range, most packets should reach their destination if the peer network is not overly large. We know that the probability of a packet reaching its destination after at most  $l$  hops through a network of  $n$  peers is:

$$\Pr(A \rightsquigarrow_l B) = \sum_{i=1}^l \frac{1}{n-3} \left(1 - \frac{1}{n-3}\right)^{i-1}$$

Which leads to a high likelihood that a packet will reach its destination before the packet is dropped (for example, given a network of 50 peers, the probability that the packet will reach its destination in at most 512 hops is greater than 99.9%).

A definite problem with the current protocol is that duplicate packets are possible, as a packet might be forwarded to the same peer (who is also the recipient) more than once. This suggests that the time-to-live range should be chosen with care, and chosen to suit the number of peers.

## 5 Implementation and Conclusion

We have presented a very simple protocol that we believe achieves anonymous connections between two peers with a high tolerance for collaborating peers in the network, with a low probability of packet loss, but at the expense of performance.

A better revision to the two-phase key exchange would be to adopt the model of OpenPGP [1] so that messages will be encrypted and authenticated on a per-peer basis, though using a symmetric key for an extended session is advantageous. The focus in this paper was on the routing layer, and it should be a simple task to adapt a scheme for privacy and authenticity to this routing layer.

A basic implementation of this protocol is being written in Java. Unfortunately, at the time of this writing, it is still incomplete. The basic routing layer is in place, but it doesn't use the cryptographic operations described above.

## 6 Future Work

An elegant alternative to the stateless random forwarding presented above would be a stateful forwarding that ensured that each packet would be sent to each peer exactly once. On receipt, each peer would insert its own identifier into a randomly-ordered set (each peer could even shuffle this set before sending it again), then forward the packet to a randomly-selected peer from that set. This solves many problems in ANET, namely that packets might arrive more than once, or not at all. Such a scheme requires a tightly-knit group of peers (adding or removing peers would not be trivial) but such a protocol does have some interesting properties, and is worthy of further study.

## References

- [1] J. Callas, L. Donnerhackle, H. Finney, and R. Thayer. RFC 2440: OpenPGP Message Format. <http://www.ietf.org/rfc/rfc2440.txt>, 1998.
- [2] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.
- [3] The Hacktivism Group. The Six/Four System. <http://www.hacktivism.com/projects/>.
- [4] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.
- [5] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, 4–7 1997.
- [6] M. Wright, M. Adler, B. Levine, and C. Shields. An analysis of the degradation of anonymous protocols, 2001.