Hordes: A Multicast Based Protocol for Anonymity*

Brian Neil Levine
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01060
brian@cs.umass.edu

Clay Shields
Dept. of Computer Science
Georgetown University
Washington, DC, 20057
clay@cs.georgetown.edu

Abstract

With widespread acceptance of the Internet as a public medium for communication and information retrieval, there has been rising concern that the personal privacy of users can be eroded by cooperating network entities. A technical solution to maintaining privacy is to provide anonymity. We present a protocol for initiator anonymity called Hordes, which uses forwarding mechanisms similar to those used in previous protocols for sending data, but is the first protocol to make use of multicast routing to anonymously receive data. We show this results in shorter transmission latencies and requires less work of the protocol participants, in terms of the messages processed. We also present a comparison of the security and anonymity of Hordes with previous protocols, using the first quantitative definition of anonymity and unlinkability. Our analysis shows that Hordes provides anonymity in a degree similar to that of Crowds and Onion Routing, but also that Hordes has numerous performance advantages.

*This study was supported in part by grants ANI-0087482, ANI-0296194, and EIA-0080119. from the National Science Foundation, and 2000-DT-CX-K001 from the U.S. Department of Justice, Office of Justice Programs. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the Department of Justice.

1 Introduction

The rapid public acceptance of the Internet as a means of communication and information dissemination is creating previously inconceivable opportunities for gathering information about individuals. This is due to the fundamental nature of the Internet Protocol (IP) that is used to communicate across the network. Each IP packet carries the IP address of the machine that sent the packet, as well as the IP address of the intended recipient of the packet. Under normal communication, any *eavesdropper*, a machine that sits on the network along the path a packet travels, can easily determine what entities are communicating, and any recipient of a packet is able to determine the source directly from received packets. While IP addresses do not necessarily uniquely identify an individual, it may be possible to link even dynamically assigned IP numbers to an individual if they access different services with the same assigned address, or if records are available about whom was assigned which address during a particular period. Such monitoring and information gathering activities by eavesdroppers or recipients of packets can adversely affect persons communicating over the Internet.

To some, it may seem at first that anonymity has only malign use. However, the benefits of anonymity outweigh any harmful consequences of its widespread availability. Anonymous communication can encourage a number of beneficial activities such as anonymous tips for investigative journalists and law enforcement officials, whistle blowing without fear of reprisals, self-help discussion groups without fear of embarrassment, and personal privacy protection in general during web browsing [19]. Several harmful abuses of anonymous communication have also been identified—including spamming, deception, and fraud—but in studying the issue, the American Association for the Advancement of Science (AAAS) has concluded that "anonymous communication online is morally neutral"; and that "anonymous communication should be regarded as a strong human right; in the United States it is also a constitutional right" [33].

The realization that some solution is needed for providing privacy and anonymity on a network is not new [12, 13, 34, 28, 27, 11], and previous work has successfully provided solutions for Internet anonymity. In this paper, we present a new protocol for providing anonymous communication on the Internet called *Hordes*, which provides a level of comparable anonymity to recent protocols [27, 28] while reducing the amount of work required of participants, as well as significantly reducing the latency of data delivery and the link utilization. Hordes achieves these reductions by making use of multicast communication, and is the first protocol designed to provide anonymity that does so. We introduce an explicitly quantitative definition of anonymity and show that Hordes maintains comparable anonymity to similar protocols at all times. Additionally, we compare the performance of Hordes and previously proposed protocols to overt communication on the Internet with network simulations.

In Section 2, we overview related work and provide a brief review of multicast routing, which Hordes uses to reduce the overhead required of participants. In Section 3, we introduce a quantitative method of comparing the anonymity provided by anonymous protocols. In Section 4, we provide a detailed description of Hordes. In Section 5, we discuss the relative performance of our technique to two past approaches. Section 6 provides an overview of the operation of existing multicast routing protocols to demonstrate their compatibility with Hordes. We offer concluding remarks in Section 7.

2 Background

In this section, we describe past work that influenced the development of Hordes and review multicast routing, a one-to-many network service that Hordes uses to provide receiver anonymity while reducing data delivery latency.

2.1 Previous Work

A common and simple solution for providing anonymity to the initiator of an Internet connection is to use a *proxy*, which is a single server that accepts connections from the *initiator* of an anonymous connection and forwards them on to the *responder*, i.e., the host that the initiator wishes to contact anonymously. With this single-proxy method, all the responders ever learn is the proxy's address; thus, the initiator is anonymous to the responder. For example, the Anonymizer [1] and the Lucent Personalized Web Assistant (LPWA) [3] provide anonymity using such methods. They also provide additional services, the Anonymizer removing identifying information from the data stream, and the LPWA maintaining a number of anonymous identities for each user. When using a proxy, the initiator is anonymous from the responder and any eavesdropper on the path from proxy to responder. While this may provide adequate anonymity in many cases, the proxy itself can determine the initiator's identity. (In this paper, the IP address of an entity is equivalent to its identity.) In situations where anonymity with respect to all entities on the path is required, some other solution is necessary. There are cryptographic solutions that provide a high degree of anonymity [13, 34], though at a high cost in terms of network traffic and processing. In this paper we concern ourselves protocols that trade security for network performance.

2.1.1 Onion Routing

Our work is compared directly against two protocols in particular. We present an overview of the operation of these protocols here, and refer the reader to the original papers for details [27, 28].

The Onion Routing protocol [27], which is based upon the idea of mixes [11], is more robust than single-proxy methods for anonymous communication. In Onion Routing, a series of proxies communicating over encrypted channels cooperate to forward data to a responder. Data is wrapped in a series of encrypted *layers* that to form an *onion*. The layers of the onion are peeled off at a series of proxies (onion routers) along a path towards the responder. Additionally, mix-type multiplexing, in which packets are reordered before being forwarded, can be employed to thwart traffic analysis.

It is assumed that each onion router knows the identities and public keys of each other onion router. The initiator, I, begins by choosing a route through the other onion routers to the responder R. For each onion router on the path, σ, the initiator constructs a layer of a connection setup packet consisting of the IP address of the next onion router, the encryption key seed information shared with the next onion router k, and the successor's layer. The innermost layer of the onion contains the identity of the responder and the data to be sent. Each layer is encrypted with the public key of the corresponding router, $K_{\sigma+}$. Each onion router pair uses a locally unique *anonymous connection* *identifier (aci)* so that subsequent communication does not require sending another onion.

$$I \rightarrow \sigma : \text{aci}, k, \{\sigma', k', \{\sigma'', k'', \{R, \text{data}\}_{K_{\sigma''+}}\}_{K_{\sigma'+}}\}_{K_{\sigma+}} \tag{2}$$

As the packet is forwarded through the path of onion routers, the layers are peeled off. When the packet reaches the last onion router in the path, the data is forwarded directly to the responder. All requests from the initiator are sent along the same path of onion routers. Replies are sent to the last onion router on the path, which in turn forwards the data along the reverse path of onion routers towards the initiator. In implementation [5], Onion Routing is not typically deployed at every host. Instead, a number of dedicated onion routers are available for use, and an initiator must connect to one of these to contact the receiver. The first onion router thus knows all initiators it is servicing. Should an onion router be corrupted, all initiators that use that router could be exposed.

2.1.2 Crowds

The Crowds protocol [28] is similar in operation to Onion Routing; however, the path through cooperating proxies called *jondos* is chosen randomly, on a hop-by-hop basis, as the initial request is forwarded through the crowd. Once a path out of the crowd is chosen, it is used for all anonymous communication from the initiator to any responder within a 24-hour period.

Crowds begins with an initialization protocol. When complete, the initiator knows a private, symmetric key between itself and every jondo in the crowd. To send data, the initiator constructs and forwards a packet containing a random path id, p; the IP address of the responder, R; and the data. All are encrypted with the key K_{Ij}, shared with the randomly chosen next jondo, j.

$$I \rightarrow j : \{R, p, \text{data}\}_{K_{Ij}} \tag{1}$$

Each crowd member receiving a packet with a new path id then randomly decides based on a *probability of forwarding*, denoted P_f and satisfying $0.5 \leq P_f < 1$, whether to forward it on to the responder or to another randomly chosen jondo. Eventually, a jondo will decide to forward the packet to the responder based on P_f.

$$j' \rightarrow R : p'', \text{data} \tag{3}$$

Once the responder receives the packet, it returns a reply packet along the reverse path of the request. Subsequent packets between the initiator and responder always follow the same path. This use of *static paths* is necessary because if a number of jondos collaborate to discover the identity of an initiator, then each new path that is formed gives them the opportunity to narrow down the identity of an initiator. The initiator must be on each path, and therefore shows up more often than any other jondo. To limit the number of paths available to collaborators, Crowds only changes paths at a set period, typically every 24 hours; this change of paths is termed a *commit*. The use of static paths can lead to a slightly different problem. If new members that join immediately create a path, they can be easily identified as new members by the first jondo they reach if most or all other crowd members have already established paths. A new member should wait until the next commit to form a path out of the crowd. At that time, each initiator must flush and recreate all existing connections through the crowd.

While the identities of the crowd members are public knowledge, responders, eavesdroppers, and other crowd members never learn which particular crowd member is the initiator, as it is not easy to determine if the a successor on a path is sending its own message or forwarding one of another member. Each member of the crowd gains anonymity at the cost of bandwidth in forwarding others communications.

2.2 Multicast Routing

A rapidly growing number of routers and hosts on the Internet are beginning to support *multicast* communication—point-to-multipoint delivery between hosts on a network. (Readers unfamiliar with the details of multicast are encouraged to see Section 6 for a more detailed overview of multicast routing and prominent multicast routing protocols.) Multicast, class-D addresses, unlike other IP addresses, do not refer to any particular device attached to the network; instead, they are essentially a label that refers to the receivers in the group as a whole, without knowledge of any particular receiver or of group membership. The number of hosts joined to a multicast routing tree as receivers, as well as their status, is dynamic and *unknown to routers and hosts.* It is these properties that make multicast useful for anonymity.

Hordes makes use of multicast communication for the reverse path of anonymous connections. This has several advantages. First, if the membership of a particular multicast group that is receiving data is determined, the actual initiator is still indistinguishable within that set as the intended recipient, as long as it is not the only member of the multicast group. Second, as a practical matter, it is difficult to determine the multicast group membership in current multicast protocols. The membership is not known to any single entity; it takes the coordination of all routers in the tree to determine the receiver set. Getting such cooperation across multiple administrative domains has proven to be difficult in the past.

Hordes is designed to place many participants in the same group for reception. Using multicast for anonymous reception thus provides anonymity in several ways. First, the destination IP address placed in reply packets is the multicast group address and not any host's IP address, and it is often difficult to determine the membership of the multicast group. This means that the host can receive data without being immediately identifiable. Second, and more importantly, if the group membership is discovered, there exists anonymity within the receiver set. While the degree of anonymity provided in the case of group membership discovery is is likely to be a lower than other existing anonymous protocols, the tradeoff is that latency and processing overhead are reduced for horde members.

In this paper, we assume that the multicast routing protocol being used creates shortest path multicast routing trees from the responder to the initiator, which is the case for the most popular multicast routing protocols in use today, e.g., PIM Single-Source-Multicast, PIM Sparse-Mode, PIM Dense-Mode, and DVMRP. Multicast is still an evolving Internet specification, and other multicast routing protocols are being developed and deployed. In Section 6, we discuss the properties of specific multicast routing protocols with regard to Hordes. The use of some of these protocols may reduce the anonymity available in Hordes, though *none* allows attacks that completely compromise its operation.

3 Defining Anonymity

Measures of anonymity proposed in previous work have been informal and do not include definitions suitable for quantitative analysis. Pfitzmann and Waidner have proposed the concept of *unlinkability* between the initiator and responder, where these two entities cannot be identified as communicating with each other, though it may be clear they are participating in some communication [34]. Syverson and Stubblebine have given epistemic characterizations of some properties of anonymity [30]. Reiter and Rubin informally defined degrees of anonymity that can exist between an initiator and responder [28], but the differences between these degrees can be vague. Here, we define anonymity precisely. It is important to note that when defining anonymity of an entity in the network, we do so *with respect to* some other single entity. We will see that anonymous protocols do not always provide uniform degrees of anonymity with respect to all entities in the network; therefore the particular entity, e, must be specified.

Let $\Pr_e(x)$ be the probability that entity x is the initiator of a connection, as determined by observations by entity e. Let x be a member of non-empty set S, where $\sum_{y \in S} \Pr_e(y) = 1$. Let $d_{x,e}(A)$ be the *degree of anonymity* provided for some entity x with respect to another entity e while using a specified anonymous protocol A, formally defined as follows:

$$d_{x,e}(A) = 1 - \Pr_e(x) = \sum_{y \in S \setminus \{x\}} \Pr_e(y) \tag{1}$$

In the protocols discussed here, participation in the protocol is equivalent to membership in S, and is public information which e could easily obtain. Each protocol attempts to make this public set of cooperative hosts appear to have equal probability of being the originator of a communication. If all members of S have an equiprobable chance of being the initiator, then $d_{x,e}(A) = 1 - 1/|S|$. However, it can be possible for e to disrupt equiprobability, for example by analyzing predecessor information over time, as discussed in Section 4.3. In examining the security of protocols in Section 5, we will assume a possibilistic approach and assume equiprobability. This is consistent with other approaches to examining anonymity [30]. We have produced other work that examines attacks against equiprobability [36].

We define the *baseline* degree of anonymity provided by a protocol for a set of collaborating entities S,

$$d(A) = \min\{d_{x,e}(A)\}, \forall e \in E, \forall x \in S \tag{2}$$

where S is the set of entities whose anonymity is maintained by the protocol A, and where E is the set of all entities in the network. When it is clear from context, we drop the reference to protocol A and simply write $d_{x,e}$.

Reiter and Rubin loosely defined several intervals of degrees of anonymity. Here we re-state their definitions and add formal probabilistic equivalences.

- *Provably Exposed:* the attacker can be certain that x is the initiator. $d_{x,e} = 0$.

- *Exposed:* there exists a possibility that x is not the initiator. $0 < d_{x,e} < \frac{1}{2}$.
- *Probable Innocence:* x appears no more likely to be the initiator than not to be the initiator, but x appears more likely than all other entities. $\frac{1}{2} \leq d_{x,e} < d_{y,e}$, for all $y \in S \setminus \{x\}$. It follows that $d_{x,e} < (1 - \frac{1}{|S|})$.
- *Beyond Suspicion:* x appears no more likely to be the initiator than any potential entity in the system. $|S| > 1$, $(1 - \frac{1}{|S|}) \leq d_{x,e}$, and $d_{y,e} \leq d_{x,e}$ for all $y \in S \setminus \{x\}$.
- *Absolute Privacy:* The attacker cannot perceive the presence of communication, and therefore cannot determine the membership of S. Let $|S| = \infty$ and $d_{x,e} = 1$.

When entity x is probably innocent at least, we say *minimal anonymity* has been achieved.

The protocols that we consider in this paper protect the anonymity of the initiator. We define the following formal requirements for initiator anonymity.

1. The initiator achieves minimal anonymity as the originator of a message *destined* for a known responder.
2. The initiator achieves minimal anonymity as the intended *recipient* of a message originated by a known responder.

These two conditions are necessary and sufficient for the provision of initiator anonymity. In the case that the initiator is exposed but the responder is not, the protocol is no longer anonymous with respect to the initiator, but it is *unlinkable* as the identity of the responder is unknown. This concept is important because even when a protocol does not provide anonymity for the initiator, it may provide unlinkability. In some situations, discovering that an entity is communicating anonymously may not be sufficient for the attacker, in which case unlinkability is a desirable property of a protocol.

The protocols explored in this paper attempt to make a set of cooperative hosts all appear to have equal degrees of anonymity. We can see that increasing the cardinality of S has diminishing returns on the degree of anonymity. For equal degrees of anonymity among a set S, $|S| = 20$ is much more anonymous than a set $|S| = 2$ entities; however, a set S with 220 entities is not significantly more anonymous than one with 202. For equiprobability, as S increases, increasingly large jumps in the size of S are required to quantitatively say a protocol has significantly increased its provided anonymity.

The best protocols for anonymity have a number of qualities. First, they do not require increased amounts of work or resources on the part of initiators and cooperative entities as the degree of anonymity increases. Second, the amount of work or resources required by an attacker to break an anonymous protocol should increase (or at least not decrease) as the degree of anonymity increases. Third, the amount of work required of attackers should be significant. Finally, third-party entities should not be trusted with the identities of initiators.

4 Hordes

Hordes employs multiple proxies similar to those used in the Crowds protocol to anonymously route a packet towards the responder, but then uses multicast services to anonymously route the reply to the initiator. Performance results presented in Section 5 demonstrate that Hordes has a little more than half the round trip latency of Crowds, does not require large routing tables at hosts, is not subject to a passive traceback attack, often uses fewer network resources, and requires less work from cooperating jondos.

Initialization. Initialization occurs in five steps. The purpose of initialization is to provide new horde members with an authenticated, "fresh" list of other horde members. First, the initiator, I, sends the server,S, a request to join the horde; included with the request is the IP address of the initiator IP_I, a nonce, N_I, and the initiator's public key K_{I+}. Notice that the limiting factor in joining the horde is the IP address; the public key may be generated just for the purposes of participating in the protocol and does not have to be part of any public key infrastructure. We do assume that each member of the horde possesses the server's public key. Such participating hosts are called *jondos* in the Crowds protocol, and for clarity we conform to this convention, though in the specification we will refer to horde members as h. The initialization message is as follows.

$$I \rightarrow S : \mathrm{IP}_I, N_I, K_{I+} \tag{1}$$

The server responds with a signed join acknowledgment that consists of a new nonce and a repetition of the initiator's nonce. The purpose of the exchange of nonces is to ensure that the protocol run is fresh, thereby limiting the ability of an attacker to replay messages to the server. It also ensures that the server can verify that a new member is joining the horde, and prevents an attacker from adding bogus members to the horde by replaying old messages.

$$S \rightarrow I : [N_I, N_S]_{K_{S-}} \tag{2}$$

The initiator checks the freshness of the nonce, and if it is valid, replies with a signed copy of the nonces. This check prevents an attacker from later replaying an old list of the members of the horde.

$$I \rightarrow S : [N_I, N_S]_{K_{I-}} \tag{3}$$

If the nonces are correct, the server sends a multicast base address M used by all horde members (explained below) and a list of all other horde members and their public keys. The server demonstrates that the list is fresh by including the nonces, and authenticates the list by signing it.

$$S \rightarrow I : [M, \mathrm{IP}_h, K_{h+}, N_I, N_S]_{K_{S-}}, \forall h \in H \tag{4}$$

The server then informs the entire horde that I has joined by sending a single multicast message. The announcement includes a timestamp, TS, to ensure the update is not a replay.

$$S \rightarrow H : [\mathrm{IP}_I, K_{I+}, TS]_{K_{S-}} \tag{5}$$

Data Transmission through the Horde. Note that since a horde initiator is sending to the responder via unicast and receiving replies via multicast, it cannot use standard TCP connections. Instead, a TCP connection between the responder and the initiator must occur encapsulated within UDP packets transferred between each jondo on the forward path, and within the UDP packets multicast to the initiator from the responder. If the responder were not aware of the Hordes protocol, the member of the horde that determined that it would send the request to the responder (the last hordes member on the forward path) could instead form a TCP connection to the responder and could then multicast the data back to the initiator.

Step 1. The initiator (and each other jondo) randomly picks a small subset of jondos $F \subset H$ used to forward messages. It then sends each forwarder f in the subset F a symmetric key, K_f, encrypted in the forwarder's public key, K_{f+}, and signed with the initiators own private key, K_{I-}. The reason for and advantages of selecting a subset of forwarders is discussed below, in section 4.3.

$$I \rightarrow \forall f \in F : \{[K_f]_{K_{f+}}\}_{K_{I-}} \tag{1}$$

Step 2. The initiator randomly picks a multicast group, m, to use to receive replies. This address is chosen by computing a range that is at minimum the base multicast address M, which is known to all horde members, and at maximum M plus a number that depends on the number of members in the group as described in Section 4.1 below. Each initiator chooses and subscribes to only one group in that range. Group address selection is discussed in detail below, but the point of picking different groups is to distribute receivers so that *receivers do not listen to all traffic.* At this point, the initiator should join the multicast group that it selected as a receiver, if it is not already a member.

To forward data, the initiator sends a message to a random jondo in its forwarding subset, $h \in F$. The message includes the address of the responder, R; a random number used later to identify a particular reply on the multicast tree, id; the multicast group on which the responder will send replies, m; and the data. The random number needs to be large enough to minimize the chance that a collision will occur if some other jondo choose to receive on the same multicast group—128 bits should be sufficient. This portion of the message is encrypted with a symmetric key K_f, and prefaced with a key identifier, i, both of which are shared with the next forwarding hop. Key identifier and key distribution between proxies is discussed in section 4.3 below.

$$I \rightarrow h : i, \{R, \mathrm{id}, m, \mathrm{data}\}_{K_f} \tag{2}$$

Step 3. When a jondo receives a message, it chooses with probability $1 - p_f$ to send it to the responder (Step 4), or with probability p_f to forward it to another jondo from its forwarding subset. We denote this successor jondo as h'. The jondo sends the same form of message as in Step 1, however, a different key identifier, i', and shared key $K_{f'}$ are used.

$$j \rightarrow h' : i', \{R, \mathrm{id}, m, \mathrm{data}\}_{K_{f'}} \tag{3}$$

Step 4. With a high probability, and after some number of hops through the horde, some jondo (h' in the notation below) will, eventually forward the message to the responder.

$$h' \rightarrow R : m, \mathrm{data}, \mathrm{id} \tag{4}$$

Step 5. The reply is sent to the multicast group m. It is prefaced with the random number, id, to identify it to the receiver for easy reception from the multicast group. The receiver should remain subscribed to the group until the next commit to limit the effects of timing attacks.

$$R \rightarrow m : \text{id,reply} \tag{5}$$

4.1 Multicast Groups

In Hordes, the amount of work a jondo performs is dependent on the number of messages it must process. This is proportional to the number of forwarding paths the jondo appears on, and the number of other jondos that choose the same multicast group on which to hear replies. Notice that receiving a message via multicast is actually cheaper computationally than forwarding a messages (as in Onion Routing and Crowds), as the jondo needs only to check the random ID to decide whether to accept or drop the packet.

In Section 5, we see that the number of jondos that choose the same multicast group for replies affects the anonymity provided by the protocol; therefore, it is possible to tradeoff between workload and anonymity. Our desire in the design of Hordes, however, is that *no horde member should ever do more work than a Crowds or Onion Routing member while always maintaining anonymity.* Accordingly, in this section we solve for m, the number of multicast groups among which horde members should be evenly split for reception of traffic. By distributing the horde members among different groups, we maintain anonymity while limiting the number of messages each member must process. The multicast addresses are chosen from a range starting at the base multicast address M, shown in Step 4 of the initialization of Hordes. As the number of members in the horde grows, the number of groups being used grows from this base.

The requirement to provide minimal anonymity bounds m from above; there should be at least two jondos in each multicast group in case of a traceback attack. If there are n hosts in a horde, then

$$n/2 \geq m \tag{1}$$

The requirement that horde members should do no more work than crowd members bounds m from below. The amount of work crowd members are subject to is dependent on the number of other paths they appear on in the crowd. The value for the amount of work has been derived by Reiter and Rubin [28], and therefore the number of jondos in each multicast group should be less than or equal to this value:

$$\frac{n}{m} \leq \frac{2}{(1-p_f)^2}\left(1+\frac{1}{n-1}\right) \tag{2}$$

$$m \geq \frac{(1-p_f)^2 n}{2\left(1+\frac{1}{n-1}\right)} \tag{3}$$

It can be shown that m is bounded correctly by the values from lines 2 and 3. However, some jondos may be collaborators, and we do not want to fill multicast groups with all collaborators but one host. To protect the reverse path, the upper bound must not include any collaborators, which gives a new bound

$$m \leq \frac{n-c}{2} \tag{4}$$

Reiter and Rubin [28] have determined that the forward path in Crowds (and thus Hordes and Onion Routing) is not secure if this inequality does not hold:

$$n \geq \frac{p_f}{p_f - \frac{1}{2}}(c+1) \tag{5}$$

Equivalently, the limit of collaborators is

$$c \leq \frac{n(p_f - \frac{1}{2})}{p_f} - 1 \tag{6}$$

where c is the number of collaborators in the session and p_f is the probability of forwarding packets inside the Crowd or Horde. Any more collaborators than allowed by the above formula, and the forward path of Hordes does not maintain anonymity. Combining Eq. 4 and 3 and then substituting Eq. 6 gives

$$\frac{(1-p_f)^2 n}{2(1+\frac{1}{n-1})} \leq m \leq \frac{n - \frac{n(p_f - \frac{1}{2})}{p_f} + 1}{2} \tag{7}$$

It is easy to show these inequalities hold for all values of n and p_f. Shown in Figure 1 is a graph of the upper and lower bounds for a quantitative representation of how much these bounds differ with increased group size. Accordingly, Hordes has adjustable amounts of work that changes with m; however, the value of m also determines the degree of anonymity of Hordes when subject to a traceback attack, as discussed below. It is easy to see from the graph that there is a large range in which Hordes provides anonymity while requiring less work, in terms of message processing, than Crowds. It is important to note that m is independent of the network delays experienced by Hordes. Even when m is set at the upper bound so that the work is comparable to Crowds, the round trip latency exhibited with Hordes remains the same, and is still significantly less than Crowds (see Section 5).

It is important to notice that the random distribution of horde members into different multicast groups can, with some small probability, result in a single hordes member listening to a multicast group. This does not result in a violation of the anonymity of the protocol, as it would require that an attacker be able to trace the multicast tree along to determine that the receiver was alone; however, this is the weaker form of anonymity that the protocol provides.

4.2 Data Transfer Considerations

The asymmetry of the forward and reverse paths in Hordes poses a challenge for providing TCP service. The forward path from initiator to responder has a higher latency and an increased chance for packet loss as compared to the reverse path. The reverse path is the shortest multicast path from the responder to the initiator, has less chance for packet loss, and may not traverse some congested areas of the network that the forward path does. This leads to a highly asymmetric path in terms of latency and packet loss. As with other asymmetric routing scenarios, if TCP is used for congestion control and error correction, it will not respond as well to congestion nor evoke slow start quickly enough. Preliminary simulations show this to be the case if TCP were to be used with Hordes, limiting the benefits of Horde's shorter overall path. Designing protocols for congestion control and reliability for this type of asymmetry is beyond the scope of this paper, although many promising results [8] may be applied in future work. Alternatively, techniques such as erasure codes [23] may be used to ensure reliability between the initiator and responder.

Because Crowds and Onion Routing paths run over several sequential TCP connections, we can expect the performance to be worse than a direct TCP connection. The independently-run TCP connections between jondos do not have difficulties managing congestion or slow start, but jondos do end up buffering data along the path towards the responder when TCP congestion control mechanisms throttle the rate between jondos. The additional memory requirements at jondos may not scale with the number of connections through the Crowd or Onion Routers, but may be an acceptable performance tradeoff.

Hordes does have an advantage over Crowds and Onion Routing for the transfer of streaming content from the responder to the initiator. Steaming data sent on the forward or reverse paths in Crowds and Onion Routing will suffer harmful effects from the jondo-to-jondo TCP connections [31]. Packets may be retransmitted unnecessarily, without regard to deadline. Similarly, the rate may be throttled unacceptably. In summary, streaming content applications rely on UDP transmission, which cannot be provided in Crowds and Onion Routing. In Hordes, because data is always sent UDP, streaming content will enjoy better performance. Moreover, data sent from responder to the initiator will be shortest path, just as a direct connection would send it, with less chances for loss than if the packet was sent through several jondos.

4.3 Forwarding Subset Selection

It may seem that each jondo could always choose the next hop towards the responder randomly from the entire set of jondos. Unfortunately, this leads to an attack against the anonymity of the protocol first identified in Crowds [28]. Since each initiator must be a predecessor to some proxy at least once on each path it creates, it will appear more often as a predecessor over multiple commits than do the randomly chosen proxies between it and the responder. Collaborators in the group can compare information about their predecessors over multiple commits, and any entity who appears more often than others is very likely the initiator of a path. Notice that with enough path changes, even one corrupted jondo can gather the information necessary to identify an initiator. As Hordes uses the same forwarding mechanism as Crowds, it is subject to the same path analysis attack. Instead of choosing a random forwarder at each step, a small subset of all possible jondos are chosen to forward all traffic to during each period. The size of the subset is chosen so that the number of forwarders in the set is on the order of the expected number of paths a jondo would be on in Crowds. The expected number of paths is computed from p_f and the number of jondos as in Crowds [28]. This information is available to each jondo. Periodically, at each daily commit, this chosen subset changes. Hordes is therefore approximately equivalent to Crowds in its resistance to this attack, as the number of jondos that see the initiator as a predecessor is the same in each protocol, and the subset changes at the same rate as paths do in Crowds.

A symmetric key is generated and sent to each of the members of the forwarding subset prior to data transmission. Data can then be encrypted in this shared key, reducing the processing costs and reducing the forwarding latency. Hordes has the same encryption processing overhead as Crowds in forwarding messages.

4.4 Onion Routing based Hordes

In this paper we have introduced a technique for improving performance of anonymous protocols by using multicast. We have concentrated on applying multicast to the Crowds protocol, and in doing so developed Hordes, which uses the same mechanisms as Crowds for constructing the forward path, but uses multicast for the return path. As we have considered the relative merits of Crowds, Hordes and Onion Routing, it has become clear that Onion Routing has the most secure path-construction mechanism. We believe that the forwarding mechanism from mixes or Onion Routing could easily be adapted for use on the Hordes forward path. While this might require more work in terms of encryption, it would increase the security and anonymity of the protocol, particularly against collaborators, as well as obviating the need for a Hordes-aware proxy to run on each server. The next section discusses some of the reasons why using Onion Routing on the forward path might be more desirable.

5 Anonymity, Security and Performance Analyses

In the first part of this section we analyze the comparative anonymity and security of Onion Routing, Crowds, and Hordes. We consider different attacks possible against each protocol and give a quantitative analysis of the anonymity provided in the face of each attack, based on the results of Section 3. In the second part of this section, we consider the network performance of Crowds, Hordes, and overt communication over the Internet.

5.1 Anonymity and Security

We consider a number of attacks against each of the three anonymous protocols. Table 1 summarizes the degree of initiator anonymity provided by each protocol in the presence of such attacks. The degree is summarized by the value of $|S|$ (from Section 3) and assuming equiprobabilities; i.e., the degree of anonymity is $d_{x,e} = 1 - \frac{1}{|S|}$.

In cases where the initiator is discovered but the responder's anonymity is maintained, and therefore unlinkability is maintained, the section is marked "U" for unlinkable. The size of S relevant to the responder's degree of anonymity relative to the local eavesdropper is also given. In this chart, n represents the number of cooperating entities participating, i.e., the number of onion routers or jondos; g represents the number of initiators listening to a particular multicast group (in Hordes we expect this value to be n/m); r represents the number of possible responders (which may

Attack	Required Resources	Onion Routing	Crowds	Hordes
1. Observing Responder		n	n	n
2. Single Protocol Member		n	n	n
3. Active Path Traceback	network access to entire path	n	1	Forward: 1 Reverse: g
4. Passive Path Traceback	access to member routing information	1	1	Forward: n Reverse: g
5. Local Eavesdropper	communication bottleneck	U : r	U : r	U : r/m
6. Local eavesdropper and on-path collaborator	bottleneck, collaborator	U : r	1	1
7. Local eavesdropper and full path of collaborators	bottleneck, many collaborators	1	1	1

Table 1: Size of S in Onion Routing, Crowds, and Hordes during various attacks described in Section 5.

equal n if the assumption is made that each initiator is communicating with exactly one distinct responder); and m represents the number of multicast groups being used for reception in Hordes.

With respect to the responder and other members of the participating group, each protocol provides an identical degree of anonymity that is proportional to the number of members in the group.

5.1.1 Path traceback

In a traceback attack, an attacker starts from a known responder and traces the path back to the initiator along the forward path or the reverse path. There are two types of traceback attacks. In an *active traceback attack*, the attacker has control over the network infrastructure and is able to follow an active and continuing stream of packets back through the network to their point of origin. If there is only one stream, this is easy. If several streams are passing through a particular host this may be more difficult, especially if the packets change form in the host, perhaps by being encrypted or re-encrypted with a different key. Crowds is subject to this attack. Since Onion Routing is based on mixes, which re-encrypt and re-order a number of packets before forwarding them, it will not be possible to identify which packets belong to which stream.

In a *passive traceback attack*, the attacker is somehow able to examine the routing state of members participating in the protocol and trace back the connection via the stored routing. In Crowds and Onion Routing, this requires an attacker with enough resources available to corrupt every host machine on the reverse path from the responder to the initiator. As each machine is corrupted, the protocol routing tables are examined to find the previous jondo on the specific path from the responder. Depending on the implementation, this information may be available for the full duration of time between commit periods because routers must keep static paths, and therefore unchanging routing tables as well. It is possible to open and close TCP connections as necessary, however, limiting the time available for traceback to that of the connection and any waiting period required by TCP to receive late arriving packets. Therefore, tracebacks of this nature are possible even when data is not flowing from the initiator to the responder.

In Hordes, the forward and reverse paths are not the same. An active traceback along the unicast forward path can be launched by attackers against a Hordes session, but only while the session is active. This is made more difficult by the fact that packets do not follow the same path through the network. A passive traceback would not succeed because Hordes does not maintain per-path routing tables. It is also possible to perform an active or passive traceback along the multicast reverse path in Hordes. In this case passive traceback (among a group of network routers) may be easier than among a group of widely distributed hosts, if much or all of the network is under the same administrative control. In either case, however, the identity of the initiator may not be immediately discernible. The multicast group being traced may have a number of receivers, hiding the identity of the initiator.

While it could be expected that such an attack would be very difficult to perform against a widely-distributed set of hosts, traceback is still a threat when considering powerful opponents. Research into network traceback is ongoing, and even though the area is in its infancy, some methods and tools have been developed to facilitate traceback of particular types of data traffic. This work has been generally motivated by the need to track network intruders, which is very similar to tracing back an anonymous connection, since intruders often try to disguise their location by logging in through a series of compromised hosts, analogous to anonymous proxies. One method [29] attempts an active traceback of the stream by comparing its contents at different points in the network. Another method implements the local eavesdropper to determine if a particular stream originates within a domain, or if it is being forwarded through the domain [39]. There are also automated methods of active and passive traceback that examine state within network routers to follow a stream back through the network [24, 10]. These types of methods could be modified to trace an anonymous connection as easily as one originating from a network intruder.

5.1.2 Collaborators

Any of the three protocols may not be safe against a group of *collaborators*, which are malicious participants in the protocol who communicate with each other to discover the identity of some initiator. In the extreme case, all but one host is a malicious collaborator, in which case any packets sent by the honest participant via Onion Routing, Crowds, or Hordes are clearly identifiable. Reiter and Rubin have provided seminal analysis of this issue [28]. They have found that in Crowds, for a crowd of size n with c malicious collaborating crowd members, if $n \geq \frac{p_f}{(p_f - 1/2)}(c+1)$, where p_f is the probability of forwarding a packet to the destination, then the initiator has at least minimal anonymity (in our terminology) with respect to the collaborators. This same analysis applies to the Hordes forward path, but not the reverse path.

It should be noted that all three protocols rely on the assumption that IP addresses are a relatively expensive resource, and that an attacker might have difficulty in obtaining enough different IP addresses to assemble a sufficiently large group of collaborators. In fact, this does not necessarily hold true. A reasonably powerful attacker might have little difficulty in obtaining an adequate number of IP addresses, or might even attack the unicast routing in order to hijack an entire range of addresses. Onion Routing has an advantage in that the initiator is able to choose its path, so that if some onion routers are known or suspected to be collaborators, the path chosen can exclude that group.

In Hordes, because the reply can be heard by any multicast receiver on the Internet, and because the reply comes without being processed by some other proxy, a timing attack is possible. As each jondo on the forward path is able to see what the reply multicast address is, a malicious jondo who is a successor to the initiator on the path can listen to that multicast address and attempt to correlate the reception of multicast data and the issuance of forward traffic on the forward path. If that time period is very small, then there is a good chance the predecessor is the initiator. This attack is made more difficult by the fact that any initiator will only send a fraction of traffic to any particular successor. A possible defense for this is to use Onion Routing for the forward path, as only the last jondo on the path would learn the multicast address being used, with a correspondingly smaller chance that a collaborator would learn it.

5.1.3 Malicious Jondos

Jondos in anonymous routing can easily launch man-in-the-middle attacks if data is not encrypted on the path from the initiator and responder. For Onion Routing, which relies on layers of encryption, this is not a problem. In Crowds, however, data is typically not encrypted in a manner that prevents intermediate jondos from examining the contents. This leads to an interesting "attack" in which an intermediate jondo inserts additional information, such as advertisements, into replies destined for the initiator. Furthermore, any key exchange protocol used between initiator and responder must itself be robust against a man-in-the-middle attack. Catching malicious jondos performing such attacks is exceptionally difficult as the purpose of the protocol is to protect the anonymity of all entities involved. Note that Hordes jondos cannot launch such an attack because the forward and reverse paths are not the same, though horde members could modify the contents of the messages on the forward path.

5.1.4 Local Eavesdropper

A local eavesdropper is an attacker that is able to monitor all network communications sent to or received by one particular protocol participant, but does not have access to the memory contents of the participant. Local eavesdroppers are difficult to defeat precisely because they can record and compare all incoming and outgoing messages. If the member sends a message that was not received, then it is clear that the member is the initiator of that message. Similarly, incoming messages that result in no outgoing messages are clearly replies for which the receiving node was the initiator.

However, in all three protocols the outgoing packet is encrypted, so it would not be clear to the local eavesdropper who the responder is (short of breaking the encryption). Therefore these protocols retain unlinkability in the presence of a local eavesdropper as long as the responder cannot be determined.

In Crowds, the local eavesdropper is only able to learn the identity of the responder from a collaborator who is a successor on the path, assuming that the packet was re-encrypted at each hop. If the packet is not re-encrypted, then any collaborator on the path could recognize the message and expose the responder. In the Crowds description, it is stated that all transmissions between jondos are encrypted using a pair-wise shared key; however the implementation section of the same paper describes a single key being distributed over the course of the path, and the message encryption remaining the same over each hop. In Onion Routing, the packet would have to be tracked through the set of onion routers (requiring the cooperation of every onion router on the path), because only the last onion router learns the identity of the responder. This is a strength of onion routing.

In Hordes, replies follow what is likely a different path and come directly from the responder via a multicast tree. This gives the eavesdropper the opportunity to learn the identity of the responder and remove the unlinkability, since if the only replies received were from some single source, the responder would be immediately apparent as that source. To protect against this scenario, horde members use shared multicast groups so that each receives and discard traffic meant for other members. This provides protection against a traceback attack, described above, as well as obscuring with which responder the monitored jondo is communicating. This method does, however, allow a collaborator to compile a list of responders (one of which is the actual responder) by monitoring the multicast group. The degree of anonymity provided by Hordes for the responder in this situation is equivalent to the number of active responders sending on that group. Assuming that each initiator communicates with a different responder and all responders are active and divided equally among all groups, then this size of S for any particular responder would be $\frac{n}{m}$, rather than n. Notice that these assumptions may be weak in some cases. If the local eavesdropper is monitoring an initiator who is receiving on a multicast group that is carrying no other initiators' traffic, the eavesdropper can determine the initiator. This is a trade-off; lower network latency is gained at the expense of degraded resistance to local eavesdroppers.

5.2 Link Utilization

While a possible concern about Hordes is that the use of multicast will result in excessive network traffic, we show through simulation that this does not happen. In fact, in most cases, using Hordes results in an overall lower link utilization than Crowds. We consider the overall link utilization to be the sum of all the links that a message and its reply have to travel on the path from initiator to responder and back.

As Hordes uses the same forwarding mechanism as Crowds, the forward path will grow in the same manner—an increase of one network diameter for each forwarding proxy. The reverse path is different however, as replies go by multicast. Though Hordes places more than one receiver in each multicast group, the link utilization does not rise in direct proportion to the number of multicast receivers because multicast messages only need be sent once over any link, and are copied at points where the path to different receivers diverges. The more receivers in the group, the greater chance of path commonality, and the less burden, in terms of link utilization, each additional receiver requires.

To consider under what conditions Hordes has a lower link utilization than Crowds, we ran a series of simulations on topologies generated by GT-ITM, a network topology generator commonly used in internetwork simulations [4, 37, 38]. The simulation provided a count of the links for direct connections, for Crowds, and for Hordes with varying policies of multicast receiver distribution.

We generated 50 transit-stub topologies of 5100 nodes each. The transit-stub model of the network was chosen as it resembles the Internet. Each model network generated consisted of a graph of nodes with weighted edges that were proportional to the distance between nodes; these weights were used to represent the latency between nodes. We then used Dijkstra's algorithm to determine the distance and best hop information for each node in the network. At the same time we determined which nodes were leaf nodes. For each probability of forwarding, which ranged from 0.50 to 0.90 in steps of 0.05, we made 50 trials in each generated graph for each anonymous group size, ranging from 100 to 1000 in steps of 100. For each trial we first chose the set of group members from the set of leaf nodes, then chose a random initiator and responder from the set of group members.

As shown in Section 4.1, it is possible to vary the number of receivers in a group while maintaining anonymity versus collaborators and still requiring less message processing of participants than Crowds. We therefore examined three policies: the *minimal receiver policy*, in which each group had only as many receivers as necessary to defeat collaborators, resulting in minimal anonymity and workload but requiring the most multicast groups; the *maximal receiver policy*, where the expected workload is the same as the maximal workload of Crowds, resulting in the minimum number of multicast groups but a larger amount of message processing and maximal anonymity; and a *midpoint receiver policy*, which used the average of the number of multicast groups of the minimal receiver and maximal receiver cases.

Our simulation shows that Hordes has lower link utilization than Crowds in most circumstances. Figures 2, 3, and 4 show the link utilization of the direct connection and of Crowds and Hordes for each of the three policies. In each of the minimal receiver and midpoint receiver cases, Hordes always has a smaller link utilization than Crowds. These figures show the link utilization increases above that of Crowds at a p_f of about 0.75 for the maximal receiver case of Hordes. What is interesting about this is that above this point horde members are actually doing more work than Crowds members. While the Hordes protocol is designed to do less work, the determination of what would constitute less work was made using an *upper bound* on what the expected amount of work a Crowds member would do. These cases fall into the area where Hordes does more work than Crowds, in terms of message processing required by members, but still does less work than the predicted upper limit. It is easy to avoid these cases by choosing a different policy that results in the use of more multicast groups, without significant loss of anonymity.

5.3 Network Latency

While both Crowds and Onion Routing [28, 27] provide initiator anonymity, they do so by increasing the delivery latency as data is forwarded through a number of proxies to the responder and then back to the initiator. Hordes, by using multicast for a direct return path, decreases the round trip time from initiator to responder significantly as compared to Crowds. Simulation results that confirm this expectation are available as an extended technical report [22]—Figure 5 shows the round trip times in a larger group of 1000 members using the simulation environment from the previous section and the results reported in the technical report. The latency problem in Crowds and Onion Routing stems from the fact that the path from initiator to responder can cross the network a number of times equal to the number of hops on the path, and that each hop has the potential of increasing the total latency by the maximum latency of any path in the network.

6 Multicast Routing

Multicast delivery of datagrams to a large receiver set is efficient because a single spanning tree is constructed across the network between a source and all receivers. A single datagram transmitted by the source traverses the tree, with copies of the datagram being made as necessary where the tree branches, so that all receivers receive a copy of the datagram. There are no special requirements for sending to a multicast group. Instead, the source simply addresses its datagram to the *multicast address* being used. In the IP architecture there are 2^{28}, or about 268 million, multicast addresses in the class D address space, ranging from 224.0.0.0 to 239.255.255.255.

Hosts wishing to receive messages multicast to a particular address *subscribe* to the group using the Internet Group Management Protocol (IGMP) [18] to contact their directly attached routers, which then join the spanning tree. The spanning tree of routers that connects a source to subscribed receivers is constructed by a multicast routing protocol; the most common protocols in use are the Distance Vector Multicast Routing Protocol (DVMRP) and Protocol Independent Multicast Sparse Mode (PIM-SM). In-depth descriptions of these protocols can be found elsewhere [32, 7, 26]. Unlike unicast routing, multicast routing requires that per-connection memory resources be maintained in each router that is part of the spanning tree in the form of a multicast routing table. It is important to note that the multicast routing table stored at each router on the path of the multicast spanning tree does not contain the list of hosts that are joined to any specific group. Rather, at each router on the tree, each interface is marked with a bit if it leads to another router in the same multicast tree. Thus, there is no "list" of what hosts are subscribed to each multicast group anywhere in the network; instead each router has only information about its directly connected neighbors on the tree. Currently, there are no mechanisms in place to determine the members of the multicast tree from a single location. Such a determination requires either the voluntary action of receivers joined to the group, or examination of the routing state of all routers that form the spanning tree[1]. Indeed, this has been identified as a problem for management of multicast routing [6] as well as protocols providing multicast transport services [21].

IP-multicast is based on an *open* service model. No mechanism restricts the hosts or users from creating a multicast group, receiving data from a group, or sending data to a group. Once a host joins a group, it receives all data sent to the group address regardless of the sender's source address. When hosts join a multicast group, no announcement is made to the other receivers or the source. (Hosts on the same subnet as the joining host would know because IGMP join messages are broadcast on the local subnet with a IP time-to-live value of 1.) Hosts can send to a multicast group without becoming a receiver; such hosts are often referred to as *non-member senders*. Multiple senders may share the same multicast address; if those sources shared a single multicast routing tree, or if they have separate trees leading to the receivers is dependent on the multicast routing protocol. Senders cannot reserve addresses or prevent another sender from choosing the same address. The number of hosts joined to a group as receivers is dynamic and unknown. The status of entities (i.e., sender, receiver, or both) is unknown. IP-multicast groups

[1] Notice that ICMP pings may not reveal the entire group membership as most hosts will not answer pings on a multicast address. Even if some hosts do answer pings, it is not sufficient to determine group membership. Also, ICMP TTL-expired messages are not sent for multicast packets because the TTL is interpreted slightly differently. Therefore, programs such as traceroute do not trace multicast trees.

are not managed.

There are many published multicast routing protocols, but only a few that are in the process of standardization by the IETF. In this section, we review such multicast routing protocols in order to determine if their operation violates any of assumptions necessary for the correct operation of Hordes. Only the Multicast Source Discovery Protocol (MSDP) [17] and Multicast Open Shortest Path First (MOSPF) [25] cause a problem for Hordes. These two protocols provide opportunities for attacks to reduce the degree of anonymity provided by Hordes from $1 - 1/n$ to $1 - m/n$. Such attacks require the cooperation of routers.

For all protocols, the Internet Group Management Protocol (IGMP) [18] is used to manage the connection between hosts (whether sources or receivers) and edge routers. IGMP is a simple protocol that sources and receivers use to subscribe to a multicast group, or even send data without subscribing to the multicast group (i.e., non-member senders). IGMP does not violate any assumptions in Hordes.

The spanning tree that connects edge routers is handled by any one the following multicast routing protocols. Multicast protocols generally construct either *source-rooted* trees (one per source in the session) or *shared* trees (one tree for all sources in the session). Signaling used to construct these trees are generally either *explicit-join* from each receiver towards the core or source; or each source *floods* data to construct the tree and receivers *prune* the tree when they are not interested.

- Core Based Trees (CBT) [9] uses *explicit-join* signaling to construct *bi-directional shared* multicast trees. CBT forms branches of the tree by joining toward a well-known *core router* in the Internet. Join requests travel only as far as the first router already joined to the tree, and in the worst case, reach the core. A join-acknowledgment traverses router-by-router back to the edge-router to confirm the join. As the core, nor any other host or router involved) ever learns the identities of sources or receivers (or even edge-routers) joined to the tree (other than their directly connected neighbors), no assumptions are violated.

- Distance Vector Multicast Routing Protocol (DVMRP) [35] is a *flood-and-prune* protocol that builds *per-source trees*. Sources flood the entire network with data. Routers without downstream hosts interested in the multicast group prune the tree pack to the source. (DVMRP is restricted to the MBone currently and is not intended for long term use over the entire Internet). Although the source's identity is overt, the receivers of the multicast group (which in hordes are the initiators) are never exposed. DVMRP builds tunnels of multicast connectivity between multicast-capable domains in the Internet, and therefore uses its own distance-vector-based unicast routing protocol to form branches of the multicast tree. The internal routing protocol does not violate any assumptions regarding Hordes.

- Multicast Open Shortest Path First (MOSPF) [25] is a multicast protocol based upon the link-state unicast routing present in domains running OSPF. MOSPF is intended for single autonomous domains only, and not inter-domain routing. Because with link-state routing each router learns the entire topology of the local network, MOSPF announces the multicast group membership to the domain and allows each router to compute per-source shortest path trees independently. Note that group membership information is flood to local routers and not data. Accordingly, MOSPF allows routers to launch an attack on Hordes initiators. By joining any multicast group that a horde group member appears on, they will receive all multicast packets that the responder transmits. However, the local router has no way of knowing if the horde member is the initiator for that responder or some other horde member that exists outside the domain. For this reason, Hordes maintains a degree of anonymity of $1 - m/n$ in this scenario because it is unknown which of the set of initiators joined to the multicast group is the intended recipient.

- Protocol Independent Multicast - Dense Mode (PIM-DM) [14] works the same as DVMRP, except it does not rely on it's own underlying unicast routing protocol. It assumes all unicast links are multicast capable. PIM-DM does not violate any assumptions regarding Hordes.

- Protocol Independent Multicast - Sparse Mode (PIM-SM) [16] is similar to PIM-DM only in name. PIM-SM is based on a core that receivers and sources explicitly join, much as in CBT. Sources send data directly to the core, which then disseminates the data on a *unidirectional shared* multicast tree, which is a tree that all sources share, but data flows only from the core. When any source transmits data above a threshold value, receivers send explicit-joins directly to the source, forming per-source trees. Receivers do not reveal their identity with this action. At no point in PIM-SM are the assumptions of Hordes violated.

- Multicast Source Discovery Protocol (MSDP) [17] is a temporary solution used to allow inter-domain multicast routing between PIM-SM domains. If receivers using PIM-SM wish to join multicast groups with sources located in remote domains, PIM-SM requires that the group-to-core mapping must be advertised to all edge-routers in PIM-SM domains. When crossing provider domains, an inter-domain multicast routing solution is required. In MSDP, neighboring domains (i.e., peers) announce sources to each other using *source active* messages. MSDP floods source information to all other PIM-cores on the Internet using TCP links between cores. Cores servicing receivers that are interested in a particular source then join the group using UDP messages sent direct to the source's edge router. Unlike every other multicast protocol, these join messages are not processed by each router, hop-by-hop, but rather only by the source's edge router. For this reason MSDP allows a source's edge router to know the PIM-core—and thus the domain—of the receiver that has joined the multicast group. This protocol does reduce the anonymity provided by Hordes: responders will see joins from domains for all horde members that have selected that group, resulting in a degree of anonymity of $1-m/n$. However, MSDP is a temporary solution with many unattractive features [15]. In addition to its unscalable flood-and-prune nature, MSDP does not allow receivers in separate domains to form shared routing trees to common sources. MSDP essentially forms dynamic tunnels through the Internet, rather than taking advantage of true multicast deployment. MSDP is to be replaced by BGMP.

- The Border Gateway Multicast Protocol (BGMP) [2, 20] is an inter-domain protocol used to manage inter-operability between multicast routing protocols in different domains. It uses bi-directional shared trees between domains and relies on core-domains of multicast groups.(In BGMP, entire domains act as cores.) At times, BGMP will form per-source trees between sources and receivers. BGMP does not violate any assumptions in the operation of Hordes.

7 Conclusions

The contributions of this paper are threefold. First, we have introduced Hordes, a new protocol for anonymous communication on the Internet, and shown how it is superior in performance and in terms of anonymity (with respect to various attackers) to previous protocols. Our method of explicitly quantifying the anonymity of a particular protocol is the second contribution, as previous methods did not provide a strict measure of the anonymity of a protocol. Third, for the first time, we have performed a detailed comparison of existing protocols.

Hordes is the first anonymous protocol to take advantage of the performance benefits and anonymity inherent to many IP multicast routing protocols. IP multicast can provide a receiver with anonymity while providing a shorter path through the network, thus reducing the latency incurred by the communication. We considered the performance of Crowds, Hordes, and overt communication using a simulation based on an Internet-type topology. This simulation gave results about the latency and link utilization of each protocol. Although still longer than overt communication, Hordes has a little more than half the round trip latency as Crowds. Hordes distributes multicast receivers among a range of multicast addresses. This limits the number of messages that any member has to process to ensure a lower workload than a Crowds member. By varying the size of the range of addresses used, Hordes can ensure that the overall link utilization in the network is less than that of the equivalent Crowd at all times.

In order to evaluate Hordes in comparison with existing protocols, we have introduced explicitly quantitative definitions of anonymity and unlinkability. Our anonymity and security evaluation concluded that Hordes maintains minimal anonymity at all times, other than for attacks for which either or both of Crowds or Onion Routing also fail. We also note that no per-session or per-initiator routing information is stored in Hordes jondos, which removes the threat of passive traceback attacks. For some attacks, although Hordes maintains greater than minimal anonymity, Crowds and Onion Routing provide higher degrees of anonymity. This is a trade-off in the use of multicast routing to reduce the network latency of communication.

References

[1] Anonymizer web site. `http://www.anonymizer.com`.

[2] BGMP working group. `http://www.ietf.org/html.charters/bgmp-charter.html`.

[3] Lucent Personalized Web Assistant. `http://www.bell-labs.com/projects/lpwa`.

[4] GT-ITM: Georgia Tech Internetwork Topology Models. `http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html`, 1996.

[5] Zero Knowledge Freedom Network. `http://www.freedom.net`, November 1999.

[6] K. Almeroth. Managing IP Multicast Traffic: A First Look at the Issues, Tools, and Challenges. In *Proc. of IP Multicast Initiative Summit*, San Jose, California, USA, February 1999.

[7] K. Almeroth. The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment. *IEEE Networks*, January 2000.

[8] H. Balakrishnan, V. Padmanabhan, and R. Katz. The Effects of Asymmetry on TCP Performance. *Proceedings of ACM Mobile Networks and Applications (MONET)*, 1999.

[9] A. Ballardie. Core Based Trees (CBT version 2) Multicast Routing – Protocol Specification. Technical Report RFC 2189, IETF, September 1997.

[10] H. Chang and D. Drew. DoSTracker. This was a publically available PERL script that attempted to trace a denial-of-service attack through a series of Cisco routers. It was released into the public domain, but later withdrawn. Copies are still available on some websites, June 1997.

[11] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[12] D. Chaum. Blind Signatures for Untraceable Payments. In D. Chaum, R. Rivest, and A. T. Sherman, editors, *Advances in Cryptology. Proc. Crypto'82*, pages 199–203. Plenum Press, 1983.

[13] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Receipient Untraceability. *Journal of Cryptography*, (1):65–75, 1988.

[14] S. Deering et al. Protocol Independent Multicast version 2 Dense Mode Specification. Technical Report draft-ietf-idmr-pim-dm-*.txt, IETF Internet Draft, August 1997.

[15] C. Diot et al. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Networks*, January/February 2000.

[16] D. Estrin et al. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. Technical Report RFC 2362, IETF, June 1998.

[17] D. Farinacci, Y. Rekhter, P. Lothberg, H. Kilmer, and J. Hall. Multicast source discovery protocol (MSDP). Technical Report draft-farinacci-msdp-*.txt, IETF Internet draft, June 1998.

[18] W. Fenner. Internet group management protocol, version 2. Internet-Draft, February 1996.

[19] R. Kling, Y.-C. Lee, A. Teich, and M. Frankel. Assessing Anonymous Communication on the Internet: Policy Deliberations. *The Information Society*, 15(2), 1999.

[20] K. Kumar et al. The MASC/BGMP Architecture for Inter-Domain Multicast Routing. In *Proc. ACM SIGCOMM 98*, September 1998.

[21] B. Levine, S. Paul, and J. Garcia-Luna-Aceves. Organizing Multicast Receivers Deterministically According to Packet-Loss Correlation. In *Proc. ACM Multimedia 98*, September 1998.

[22] B. Levine and C. Shields. A Protocol for Anonymous Communication Over the Internet. Technical Report 00-26, University of Massachusetts, Amherst, Department of Computer Science, May 2000.

[23] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and V. Stemann. Practical Loss-Resilient Codes. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC 97)*, 1997.

[24] G. Mansfield, K. Ohta, Y. Takei, N. Kato, and Y. Nemoto. Towards Trapping Wily Intruders in the Large. In *Proceedings of the Second Annual Workshop in Recent Advances in Intrusion Detection (RAID)*, West Lafayette, IN, September 1999.

[25] J. Moy. Multicast Extensions to OSPF. Technical Report RFC 1584, IETF, March 1994.

[26] S. Paul. *Multicasting on the Internet and Its Applications.* Kluwer Academic Publishers, 1998.

[27] M. Reed, P. Syverson, and D. Goldschlag. Proxies for anonymous routing. In *12th Annual Computer Security Applications Conference*, pages 95–104. IEEE, December 1995.

[28] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[29] S. Staniford-Chen and L. Heberlein. Holding Intruders Accountable on the Internet. In *Proc. of the 1995 IEEE Symposium on Security and Privacy*, pages 39–49, Oakland, CA, May 1995.

[30] P. Syverson and S. Stubblebine. Group Principals and the Formalization of Anonymity. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99—Formal Methods, Volume I*, volume 1708 of *Lecture Notes in Computer Science*, pages 814–833. Springer, 1999.

[31] W. Tan and A. Zakhor. Real-time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol. *IEEE Trans. Multimedia*, 1(2):172–186, June 1999.

[32] A. Tanenbaum. *Computer Networks.* Prentice Hall, third edition, 1996.

[33] A. Teich, M. Frankel, R. Kling, and Y.-C. Lee. Anonymous Communication Policies for the Internet: Results and Recommendations of the AAAS Conference. *The Information Society*, 15(2), 1999.

[34] M. Waidner and B. Pfitzmann. The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability. In *Eurocrypt '89*, 1989.

[35] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. Request for Comments 1075, November 1988.

[36] M. Wright, M. Adler, B. N. Levine, and C. Shields. An Analysis of the Degradation of Anonymous Protocols. In *Network and Distributed Security Symposium*, San Diego, February 2002. To appear.

[37] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom '96*, San Francisco, CA, 1996.

[38] E. Zegura, K. Calvert, and M. Donahoo. A Quantitative Comparison of Graph-Based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, Dec. 1997.

[39] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proceedings of the 9th USENIX Security Symposium*, Denver, CO, August 2000.

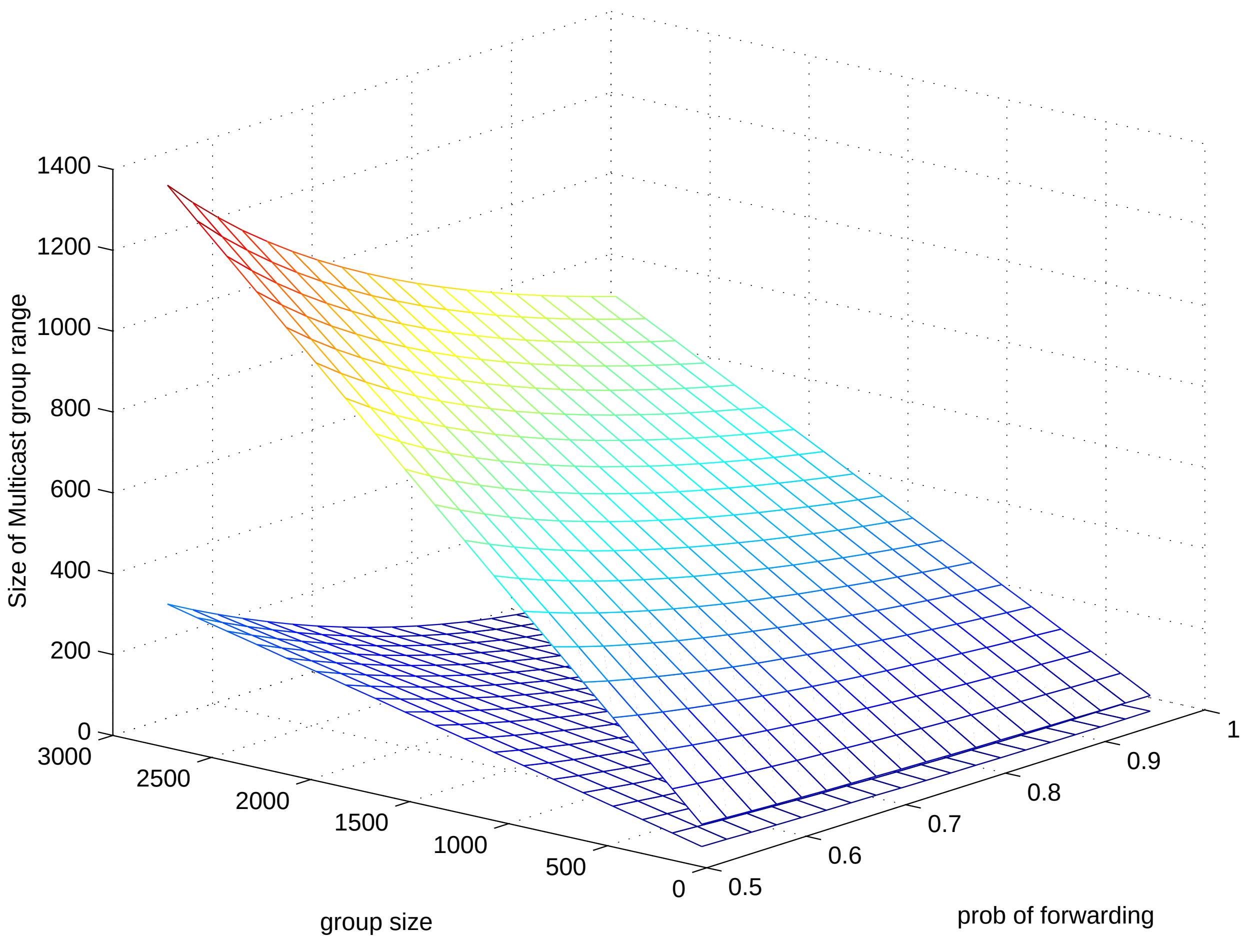

Figure 1: Lower plane: smallest size multicast group range required for minimal anonymity. Upper plane: groups used for equivalent amount of message processing by a jondo in the Crowds protocol. Between: greater-than-minimal anonymity with less work than Crowds.

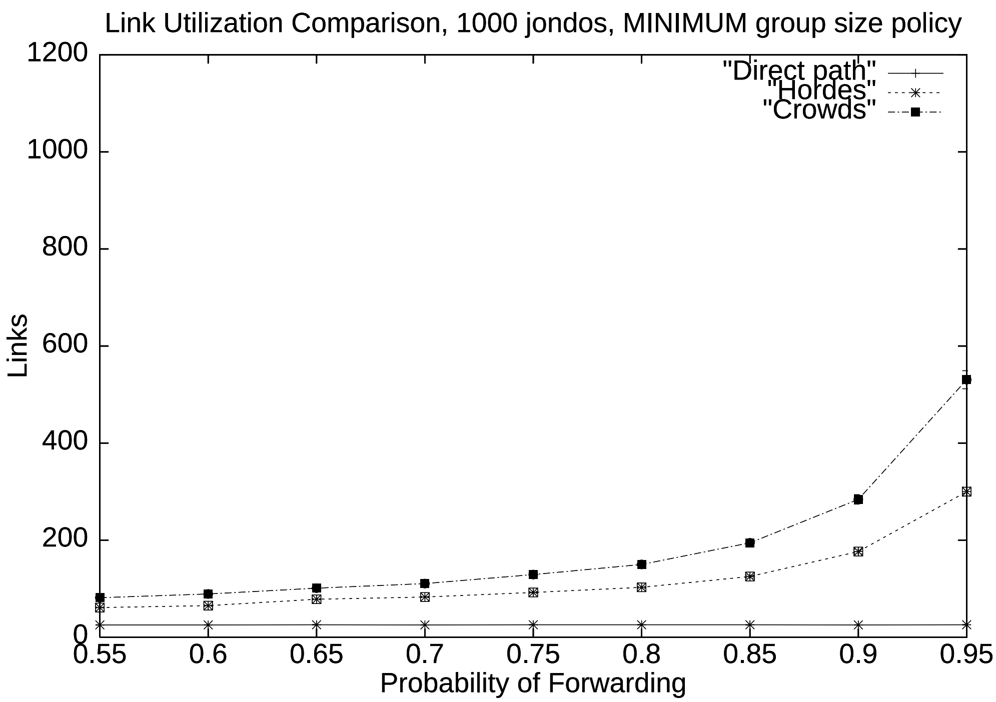

Figure 2: Link utilization comparison for the minimum policy.

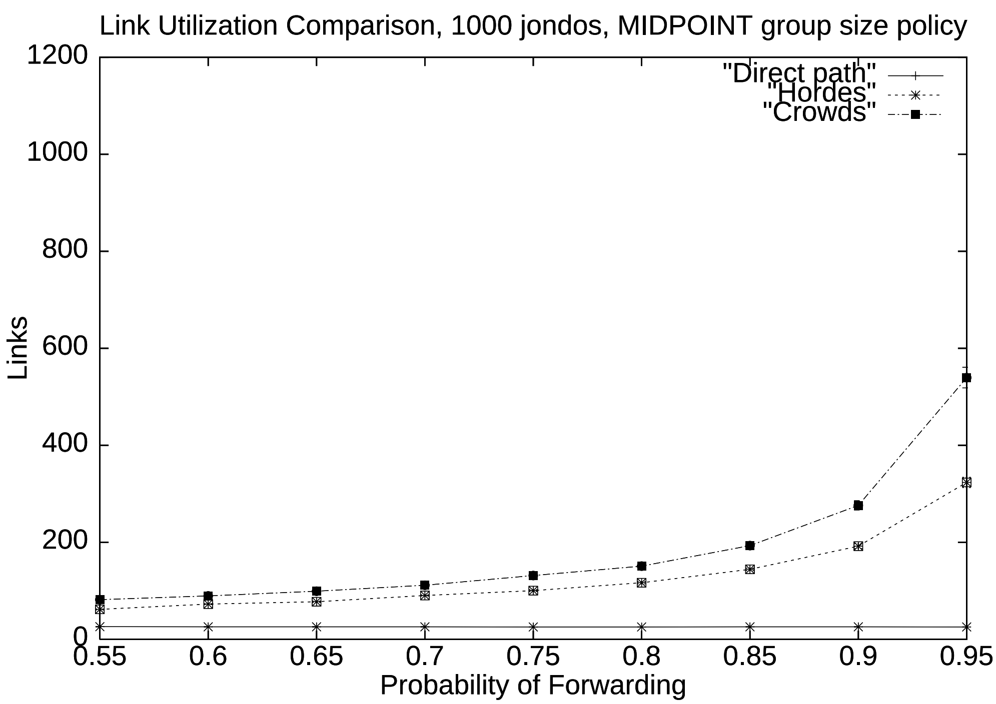

Figure 3: Link utilization comparison for the midpoint policy.

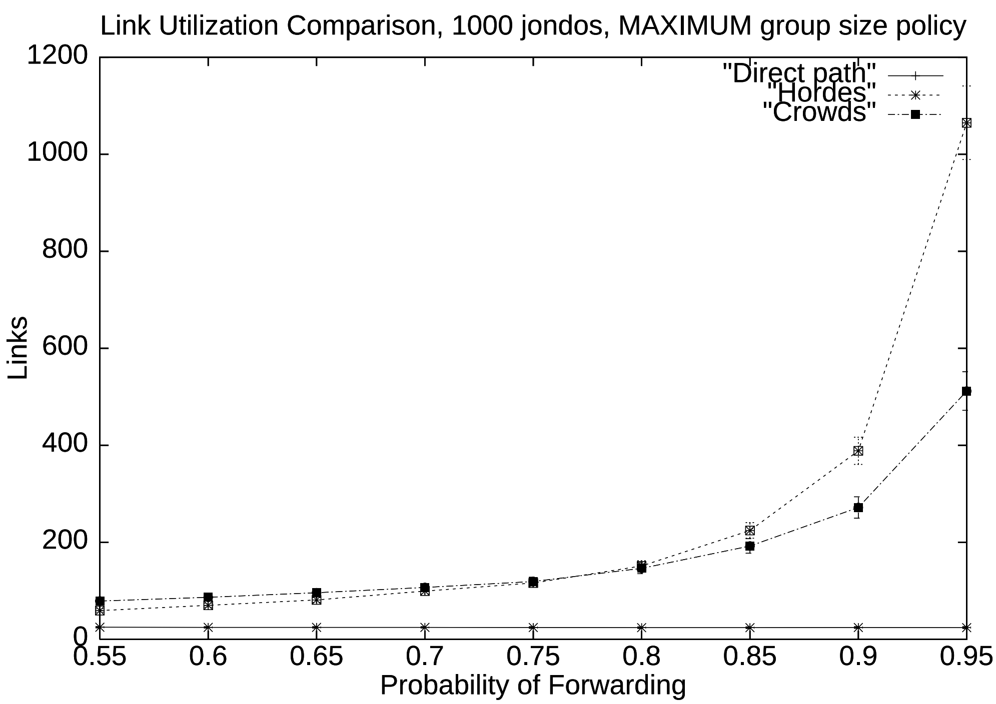

Figure 4: Link utilization comparison for the maximal policy.

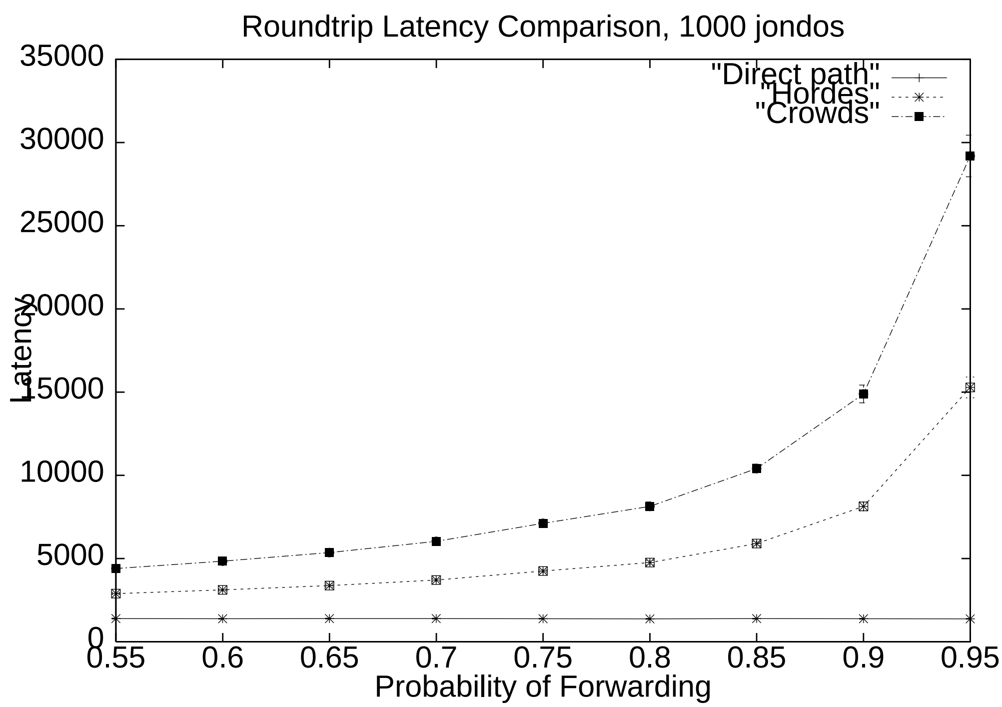

Figure 5: Roundtrip latency for varying forwarding probabilities with 1000 members.