Pssst, Over Here: Communicating Without Fixed Infrastructure

Tom Callahan†, Mark Allman‡, Michael Rabinovich†
†Case Western Reserve University, ‡International Computer Science Institute

***Abstract*—This paper discusses a way to communicate without relying on fixed infrastructure at some central hub. This can be useful for bootstrapping loosely connected peer-to-peer systems, as well as for circumventing egregious policy-based blocking (e.g., for censorship purposes). Our techniques leverage the caching and aging properties of DNS records to create a covert channel of sorts that can be used to store ephemeral information. The only requirement imposed on the actors wishing to publish and/or retrieve this information is that they share a secret that only manifests outside the system and is never directly encoded within the network itself. We conduct several experiments that illustrate the efficacy of our techniques to exchange an IP address that is presumed to be a rendezvous point for future communication.**

I. Introduction

The Internet has increasingly moved from a system used to disseminate information to users from a relatively small number of content providers to a system that facilitates sharing information among users. This style can be plainly found in the most popular destinations and applications: Twitter, Facebook, Flickr, Skype, BitTorrent, one-click file sharing systems (e.g., RapidShare), etc. The shift from merely consuming information to sharing information has in fact led to several efforts to change the basic model of networking from host-based to content-based [1], [2] as this latter has become the basic mode of operations for users. That is, users fundamentally do not want to access some host in the network, but rather want to swap a given piece of information. The techniques explored in this paper strive to transfer small amounts of information using a scheme that is not fundamentally host-centric.

In a network model where information is generally disseminated upon request, we can readily build highly robust systems. A user interested in buying a book can easily find a book seller using a well-known DNS name (e.g., "amazon.com"). Further, server farms, content delivery networks, replicated DNS servers, geographically disparate replicas, multi-homed connectivity, etc. provide robustness of operation. We refer to this as the *central hub* model. Even if physically distributed, the service is orchestrated at some easy-to-find and highly robust central location. This model makes perfect sense for certain activities (e.g., legitimate e-commerce).

However, as noted above, users have evolved to become the most prolific content providers on the Internet. In technological terms this shift has manifested in one of two basic ways: (*i*) using a central hub to connect users and hold the shared content (e.g., Twitter) or (*ii*) using a central hub as a bootstrapping mechanism for direct peer-to-peer information exchange (e.g., a BitTorrent tracker or, in the trackerless variant, a site listing an existing DHT node). While the role of the central hub is reduced in the second approach, it is still required. Although a lightweight central hub may be perfectly reasonable in some cases, there may be other cases where such a central presence is undesirable, such as:

- For peer-to-peer systems, requiring a central hub to bootstrap establishes a system vulnerability that can hamper operation even though the major functionality is distributed at the peers. For instance, if the central hub loses connectivity (power, etc.) the larger system would likely be still functional if not for the inability to bootstrap. Therefore, for robustness reasons, not depending on a fixed central hub is useful.
- Another aspect of using a central hub is that it provides a tangible choke point that can be readily blocked by policy. For instance, blocking a large BitTorrent tracker could affect many peers even though the peers themselves do most of the work to exchange files independently from the tracker once bootstrapped. Another example is the recent case of Egypt disconnecting its major ISPs from the broader Internet—which effectively disconnects users from myriad central hubs. However, if local connectivity remains, users could in principle bootstrap to communicate locally even though their usual means for doing so is disrupted.

This situation begs a question: *Can we increase robustness and flexibility of information sharing services by allowing consensual actors to initiate communications over the network without a central hub?*

Within a small area this is straightforward. For instance, within a broadcast domain one party could encrypt a message with a secret that was pre-arranged with the recipient(s) and then broadcast the message. The intended recipient(s) would be the only ones who could make sense out of the message. Further, there is no direct targeting of the recipient(s). While such a scheme is trivially possible it does not address our question when we scale beyond individual broadcast domains. However, this limited scheme provides for a model of sorts for solving the problem in a broader way.

In this paper we develop a global *covert broadcast domain* that allows actors with only a simple shared secret to exchange small messages without the secret ever being directly used within the network (and thus itself becoming a central hub, of sorts). This message could be self-contained information or a way to bootstrap further communication. We develop this covert broadcast domain by using standard DNS servers to hold information not in the traditional sense of serving records, but by leveraging the caching behavior of the servers to convey information. Further, the scheme is not dependent on any particular DNS server, but rather any DNS server the actors agree on (as discussed in § II). In other words, we design a technique that factors out the need for a well-defined central hub for information sharing and/or bootstrapping.

We explain the mechanism in detail in subsequent sections. However, as a touchstone the reader can think of breaking a message into its component bits. Each bit is represented by a cached record in an arbitrary DNS server the actors have agreed upon. The value of each bit is represented by the returned TTL value of the DNS record—e.g., the one bits may have a TTL 10 seconds larger than the zero bits. In this way we use DNS servers' natural capabilities of caching and aging records to encode ephemeral information in the system without relying on any particular fixed infrastructure or name. In the remainder of this paper we show we can accurately publish and query for such information.

Note: An extended version of this paper that includes additional experiments and discussion is available [3].

II. DNS Server Discovery

As sketched above, we leverage caching DNS servers to hold information. To fulfill our vision, the first premise is that there should be many DNS servers on the Internet that will hold the messages we seek to store. Further, actors should be able to independently discover common DNS servers to hold the exchanged messages. Therefore, before we embark on storing and retrieving information from DNS servers we perform a DNS scanning experiment to understand the prevalence of usable DNS servers.

Actors wishing to exchange messages must share a secret S. This is used in a number of the tasks in our overall procedure, and in particular for finding common DNS servers. While we consistently refer to S as a "secret", we note that nothing compels the communicating actors to keep S strictly private. Rather, S must be shared and S is only needed by the endpoints of the communication and not the DNS servers. For example, a BitTorrent application could maintain a hard-coded collection of shared secrets for client use. From S we define a generator function as:

$$G(x) = sha1(sha1(S) + x), \tag{1}$$

where "+" denotes the append operation and x is a string. By running $G("IP1")$ and using the low-order 32 bits as an IP address any actor holding S can independently derive the same series of hosts—by replacing "1" in the call to $G()$ with linearly increasing integers—to find a usable DNS servers to mediate their communication. Each host in the common list is probed with a DNS request for a name within a domain that we know to enable wildcards[1].

[1]In particular the name we use is "dns.research.project.visit.dns-scan.icir.org.if.problematic.HASH.ws" to be up-front about what we are doing should our experimental queries trip alarms.

In our scanning experiment we probed from roughly 80 PlanetLab nodes[2]—each using its own random S—at a rate of 2 DNS queries/second. A correct response from a given host is used to trigger two more queries of the same server to ensure the TTL is being decreased on subsequent retrievals.[3] Assuming the TTL is being correctly decremented, we consider the server to be usable. However, as we discuss in subsequent sections, it is not unusual for a DNS server to pass this initial set of checks only to display non-conforming TTL behavior during message publication.

Across 22.7 million probes we find that the hit-rate is approximately 0.4%. The median number of probes sent between identifying subsequent servers is 194, while the mean is 281. Further, we find the maximum probes sent before identifying a server is nearly 9,000, with the 99^{th} percentile being 1,284. These results make probing tractable for our purposes because even scanning at a low rate will turn up multiple servers. E.g., sending one probe per minute over the course of one day will yield five DNS servers on average. Further, once the set of servers is obtained, it can be maintained with even lower-rate probes over longer time scale. In addition, DNS servers that simply disappear (as has been noted elsewhere in the literature [4], [5]) will be readily detected as attempts are made to store information at the given server. Such knowledge can also be used to trigger a new server detection phase.

While in this paper we use relatively low rate scans for all our experiments—at most 10 queries/second—we note that using a higher scanning rate could be possible in some circumstances and allow us to find a large number of usable DNS servers quite rapidly. For instance, we conducted a small experiment that was able to identify 60 recursive DNS servers within 15 seconds using a residential cable modem connection.

Finally, we note that hit-list scanning may be more effective than random scanning. This is discussed in more detail in [3].

III. A Basic Bit Channel

As described above, our goal is to utilize DNS servers' natural ability to cache and age information to store small messages without directly inserting records into the DNS system. As an initial use case, we consider publishing a 32-bit IPv4 address using this system as a basic bit channel. We start with this use case because a bit pipe is the most basic communication channel. Further, we presume that once known, an IP address can be used to form the basis of higher layer communication. In this section we use the procedure outlined above in § II to find suitable recursive DNS servers and, as they are found, publish and retrieve 32-bit messages.

A. Procedure

The process of storing messages in DNS servers starts with a pre-arranged secret S between all parties involved in the communication. Using this secret, we define a generator as

[2]PlanetLab often experiences node churn and, while we tried to choose reliable nodes, the number of nodes used in each individual test throughout the paper varies slightly.

[3]We found in early experiments that some DNS servers do not decrement the TTL of their cached records, leading to this test.

shown in equation 1. We also need a domain we know to support wildcard DNS queries, that is, queries for unknown names within the domain still return some record. As will become clear, we also need the domain to assign a sufficiently large TTL to its DNS responses. Domains supporting wildcards are widespread [6], and we found that many also return TTLs sufficient for our needs. In all our experiments we use the ".ws" domain (an arbitrary choice that returns TTLs of 3 hours). Note, we consider alternate designs that do not have this requirement in [3].

Let M be the message we wish to transmit and M_i be its i^{th} bit. We now outline two procedures for encoding M within a DNS server D.

TTL Method: The first method we employ is based on inserting records corresponding to all bits in M in such a way that the zeros and ones are distinguishable by the TTLs returned in lookup responses after publication. The publication process proceeds as follows:

1) We generate a name for each bit M_i of the message using:
$$R_i = G(\text{"}Record\%d\text{"}, i), \tag{2}$$
where "Record" is just an arbitrary identifier that all actors involved know (here and in the rest of the paper we use a `printf`-like notation to compose strings).
2) Similarly, we generate a "barrier record" using:
$$B = G(\text{“}Barrier'') \tag{3}$$
3) Next we form sets of bit numbers, Z and U, where i is inserted into Z if $M_i = 0$ and U otherwise.
4) For each $j \in U$ we execute a DNS request to D for the hostname "R_j.ws", retrying until a response is received for each record.
5) We next pause for roughly five seconds. The choice of five seconds is arbitrary. The value needs to be more than one second as that is the granularity of DNS' TTL. We leave optimizing the publication time as future work.
6) We then execute a DNS request for "B.ws", retrying if necessary.
7) We again pause for roughly five seconds.
8) For each $k \in Z$ we execute a DNS request to D for the hostname "R_k.ws", retrying until a response is received for each record.

The general idea behind this process is that D will cache the requested records with associated TTLs that originate from the authoritative server. Given the publication pattern all R_i records that have a TTL shorter than that of the barrier record B correspond to $M_i = 1$ and records having TTLs longer than that of B correspond to $M_i = 0$.

The data retrieval process borrows steps 1 and 2 from the procedure outlined above. We then query for "B.ws" and each record in R, recording the TTLs for each returned record as B^T and R_i^T. We then set M'_i to one if $R_i^T < B^T$ and zero otherwise. At this point the retrieved M' should be equivalent to the message M that was published.

Recursion-Desired Method: The second method was sketched in a Black Hat presentation [7]. To our knowledge, there has never been any experimentation to determine whether the scheme works or how effective it may be. The procedure—which we denote the "RD method"—works as follows:

1) We generate a name for each bit M_i of the message using:
$$R_i = G(\text{"}Record\%d\text{"}, i), \tag{4}$$
2) Next we form a set U, where i is inserted into U if $M_i = 1$.
3) For each $j \in U$ we execute a recursive DNS request to D for the hostname "R_j.ws", retrying until a response is received for each record.

The general idea behind this approach is that the records corresponding to the one bits in M are cached by D, whereas the zero bits are not encoded in D in any way. We leverage this when retrieving the data by sending queries for each of the 32 names in R with DNS' "recursion desired" flag set to false. This indicates that D should only look in its own cache for the given name and not recurse up the DNS hierarchy to resolve the given name. We initialize an M' to all zeros and then any "R_j.ws" query that returns a valid response indicates that M'_j should be set to one. After considering each of the 32 records M' should be equivalent to the original M.

	TTL Method	RD Method
Pub. Attempts	125K	87K
Unusable Servers	21K	12.9K
Non-Responsive Servers	742	520
Non-Recursive Servers	2.6K	1.7K
Non-Decrementing TTL	15K	9.8K
Weird TTL Decrementing	2.8K	898
Ignores RD=0	N/A	2.6K
Usable Servers	104K	72K
Successful	92K	58.8K
Failure: Packet Loss	3.6K	4.8K
Failure: No Data Found	3.6K	3.3K
Failure: Corrupt Data	5.0K	5.4K

TABLE I
BIT-PIPE PUBLICATION RESULTS

B. Results

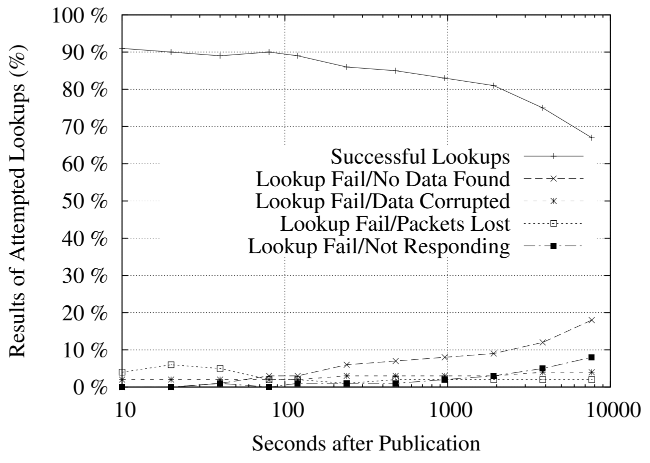
We tested the accuracy of both publication mechanisms described above by storing a 32-bit message (a la an IPv4 address for bootstrapping) in a DNS server and then attempting to retrieve the message. We use the procedure outlined in § II to probe for DNS servers and upon finding each such server we publish a message and then attempt to retrieve it. Each publication strategy is tested in its own scanning pass (which run in sequence, not parallel). We use roughly 80 PlanetLab nodes for the test. Each node performs independent scans to identify DNS servers and start the subsequent tests.

After each publication the host waits 10 seconds and then retrieves the message to assess the efficacy of the data insertion process. Table I shows the results of our publication attempts. First we note that in spite of our efforts (sketched in § II) to identify unusable DNS servers during the scanning phase we still ended up with problems in roughly 15% of the servers we tried. The largest problems come from TTL decrementing issues—even though we tried to weed out servers with this issue during our scanning pass (§ II)[4]. We note that when using the RD method, we find servers that ignore the RD=0 setting in our requests. This would be trivial to also exclude in the scanning phase and in future efforts based on the RD method we would certainly do so. While these problems do not speak to the efficacy of our information sharing technique, they do illustrate that one must exercise some care in choosing suitable DNS servers.

The lower portion of the table shows the results for usable servers. We find a publication success rate of roughly 88% for the TTL method and 81% for the RD method. The largest and most problematic publication issue is data corruption. Whereas the other issues listed in the table—no data found and packet loss—can be readily identified during the retrieval process, corrupted data gives no outward signs of problems. As we have designed a generic bit pipe, it would be possible to apply Forward Error Corruption (FEC), or simply a parity bit, to the bit-stream to reduce the number of corruption errors (at the expense of requiring more bits, of course). Also, we note that packet loss is an issue—even though we re-try pending queries every two seconds until we receive a response or have transmitted four queries. The final problem of no data being found likely comes from DNS servers that recursively lookup records, but do not cache names (for long).

We next turn to a more general investigation of information retrieval. For the successfully published messages we scheduled retrievals from a set of 55 different PlanetLab nodes chosen via round robin across our list of roughly 80 PlanetLab nodes. We schedule five retrievals (from different nodes) at each of eleven intervals between 10 seconds and 128 minutes after publication. This methodology allows us to test both (i) whether the information is available to a breadth of hosts around the network and (ii) the storage longevity we can expect from the mechanisms.

Figure 1 shows the results of retrieving the information we published as a function of time since publication. We note that just after publication the TTL method shows a roughly 90% retrieval success rate, whereas the Recursion Desired method is nearly flawless. The success rate two hours after publication drops to roughly 70% for both methods. As shown in the plot, the predominant cause of the dropoff in success is an increase in the instances of not finding the data on the server as the time since publication increases. This is a natural result of names being evicted from the cache. Even though the names nominally have TTL left, the names are to a large extent not being used and so it is natural that some LRU-like policy would evict the names corresponding to all our queries. We also note that failures to contact the DNS server rise with the time since publication, likely due to the transient nature of many of these DNS servers. The remainder of the failure causes remain fairly constant and relatively small across the



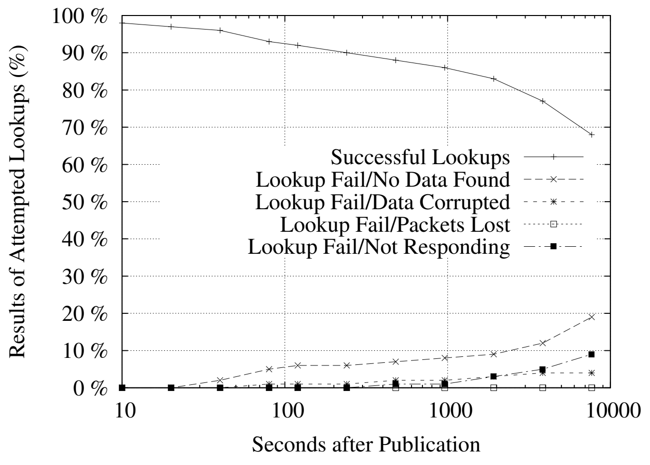


Fig. 1. Bit pipe retrieval results for the TTL (top) and Recursion Desired methods.

time period.

Finally, to ensure that the PlanetLab platform itself was not biasing our results in some fashion we replicated the above experiments from a host at ICSI. While the PlanetLab retrievals were conducted from 55 disparate machines we used the same machine at ICSI for all aspects of our tests (scanning, publication and retrieval). The results from the ICSI runs are consistent with the PlanetLab experiments. The retrieval results are similar with the RD method showing higher success soon after publication than the TTL method, but both dropping off over time. The predominant cause of failures over time is finding no data on the DNS servers, just as we find in the PlanetLab results. Therefore, overall we conclude that the PlanetLab platform itself is not significantly biasing the conclusions we draw from our experiments.

C. Discussion

We now briefly touch on additional ways to enhance the basic bit pipe we have constructed.

Robustness: A traditional way to make a bit channel more robust is to add coding to the message. For instance, a Reed-

[4]Non-conforming TTL handling does not render a server unusable in the case of RD method; however, we wanted to compare both methods over a similarly-selected set of DNS servers.

Solomon code that doubled the size of the message (to 64-bits) could detect any bit error and correct up to 16 bit errors in the message. For corrupted retrievals, we find that the corruption rate is less than half the message across both our methods at both mean and median. Coding would also help reduce the impact of losses in our results.

Widening: A natural way to widen the channel in the TTL method is to add more barrier records, which allows for more symbols to be transmitted. For instance, using two barrier records we could encode three symbols—enough to encode messages in Morse Code (using dots, dashes and spaces). We explore this further in [3]. The RD method is not directly amenable to widening due to the reliance on a fundamentally binary property of the system (namely the RD flag).

Synchronization: Note that messages in the system have a higher probability of being successfully retrieved within the first several minutes after publication. While retrievals further out in time have reasonable success rates, it may behoove some uses of such a channel to roughly synchronize publication and retrieval. For instance, when swapping S out-of-band, actors may also agree that publications will take place at the top of every hour. Even with imperfectly synchronized clocks this could increase the chances of successful message transmission.

Collisions: One issue with a single secret is that if multiple actors are publishing to that secret they will corrupt each other's messages. A straightforward way to deal with this is to assign roles to actors with respect to a particular secret during the secret exchange. For instance, for some secret S_1 Alice may be designated as the publisher and Bob the recipient, while the opposite could hold for a second secret S_2.

IV. Additional Considerations

There are additional issues that must be considered when the mechanism we outline in this paper. Due to space constraints we are only able to point interested readers to our longer technical report [3] for a discussion of and mitigations for protecting messages from eavesdropping, for mitigating the detection of use of these schemes by network monitors and additional channels for encoding information in ways similar to those described in this paper.

V. Related Work

Due to space constraints we elide a full discussion of related work. We note that the only directly related work we are aware of is [7], as discussed in detail in § III. There are a number of additional papers in the literature that are somewhat related [8], [9], [10], [11], [12]. A longer discussion of these is available in our extended paper [3].

VI. Conclusions

This paper discusses a way to communicate through the network without relying on fixed infrastructure at some central hub. This can be useful for bootstrapping loosely connected peer-to-peer systems, as well as for circumventing egregious policy-based blocking (i.e., censorship). Our techniques leverage the caching and aging properties of DNS records to create a covert channel of sorts that can be used to store ephemeral information. The only requirement imposed on the actors wishing to publish and/or retrieve this information is that they share a secret that only manifests outside the system and is never directly encoded within the network itself. Crucially we piece together a communication fabric from *classes of services* and not from particular instances of those classes. That is, while we require a recursive DNS server, we have shown that there are many DNS servers on the network that will suffice. Additionally, we require domains that offer wildcard DNS entries—which again are widespread. While the process can be optimized by assuming the actors share more than just a simple secret—e.g., hit lists to scan in lieu of random scanning, or roughly synchronizing when messages will be published—we show in this paper that these are not strictly necessary. We show that we are able to effectively use these channels for bootstrapping information. Additionally, our extended version of this paper shows that we are able to send short Twitter-like 140 character messages using a technique similar to that outlined in this paper [3]. Future work includes optimizing the process and looking for ways to make it more robust.

VII. Acknowledgments

This work benefits from discussions with Vern Paxson and Nicholas Weaver and the comments from the anonymous reviewers. This work was supported in part by NSF grants CNS-0433702, CNS-0831821, CNS-0831535 and CNS-0831780.

References

[1] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica, "A Data-Oriented (And Beyond) Network Architecture," *ACM SIGCOMM*, 2007.

[2] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking Named Content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[3] T. Callahan, M. Allman, and M. Rabinovich, "Pssst, Over Here: Communicating Without Fixed Infrastructure ," International Computer Science Institute, Tech. Rep. TR-2012-002, Jan. 2012.

[4] D. Dagon, N. Provos, C. Lee, and W. Lee, "Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority," in *Proc. of Network and Distributed Security Symposium*, 2008.

[5] D. Leonard and D. Loguinov, "Demystifying Service Discovery: Implementing an Internet-Wide Scanner," in *Proceedings of the 10th annual conference on Internet measurement*. ACM, 2010, pp. 109–122.

[6] A. Kalafut, M. Gupta, P.Rattadilok, and P. Patel, "Surveying Wildcard Usage Among the Good, the Bad, and the Ugly," in *SecureComm*, 2010.

[7] D. Kaminsky, "Black Ops of DNS," Black Hat Briefings, 2004.

[8] S. Burnett, N. Feamster, and S. Vempala, "Chipping Away at Censorship Firewalls With User-Generated Content," in *Proc. 19th USENIX Security Symposium, Washington, DC*, 2010.

[9] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *Proc. USENIX Security Symposium*, 2004.

[10] J. Dinger and O. Waldhorst, "Decentralized Bootstrapping of P2P Systems: A Practical View," *NETWORKING 2009*, pp. 703–715, 2009.

[11] C. Dickey and C. Grothoff, "Bootstrapping of Peer-to-Peer Networks," in *Applications and the Internet, 2008. SAINT 2008. International Symposium on*. IEEE, 2008, pp. 205–208.

[12] D. Wolinsky, P. St Juste, P. Boykin, and R. Figueiredo, "Addressing the P2P Bootstrap Problem for Small Overlay Networks," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*. IEEE, 2010, pp. 1–10.