Dissent in Numbers: Making Strong Anonymity Scale

David Isaac Wolinsky, Henry Corrigan-Gibbs, and Bryan Ford
Yale University

Aaron Johnson
U.S. Naval Research Laboratory

Abstract

Current anonymous communication systems make a trade-off between weak anonymity among many nodes, via onion routing, and strong anonymity among few nodes, via DC-nets. We develop novel techniques in Dissent, a practical group anonymity system, to increase by over two orders of magnitude the scalability of strong, traffic analysis resistant approaches. Dissent derives its scalability from a client/server architecture, in which many unreliable clients depend on a smaller and more robust, but administratively decentralized, set of servers. Clients trust only that *at least one* server in the set is honest, but *need not know or choose which server to trust.* Unlike the quadratic costs of prior peer-to-peer DC-nets schemes, Dissent's client/server design makes communication and processing costs linear in the number of clients, and hence in anonymity set size. Further, Dissent's servers can unilaterally ensure progress, even if clients respond slowly or disconnect at arbitrary times, ensuring robustness against client churn, tail latencies, and DoS attacks. On DeterLab, Dissent scales to 5,000 online participants with latencies as low as 600 milliseconds for 600-client groups. An anonymous Web browsing application also shows that Dissent's performance suffices for interactive communication within smaller local-area groups.

1 Introduction

Anonymous communication is a fundamental component of democratic culture and critical to freedom of speech [5, 40,56,57,59], as an AAAS conference in 1997 concluded:

> "Anonymous Communication Should Be Regarded as a Strong Human Right; In the United States It Is Also a Constitutional Right" [56]

The Arab Spring underscored the importance of this right, as organizers used pseudonymous Facebook and Twitter accounts to coordinate protests [46], despite violating those sites' Terms of Service and risking account closure [48]. Authoritarian states routinely monitor and censor Internet communication [22]: though citizens may risk "slap-on-the-wrist" punishments like blocking or throttling if detected to be using anonymity or circumvention tools [24,25,50,62], users discussing the wrong topic *without* protecting their identity risk jail or worse.

Even in countries with strong free speech traditions, anonymity can protect minority groups from discrimination [53]. Increasingly pervasive, profit-motivated tracking practices [51] have made communication linkability a widespread privacy concern [30]. Finally, anonymity plays other well-established roles in modern societies, such as in voting [2, 19, 44] and auctions [52].

Anonymous relay tools such as Tor [26] offer the strongest practical identity protection currently available, but exhibit several classes of weaknesses. First, relay systems are vulnerable to traffic analysis [6, 13, 37, 43, 45]. A state-controlled ISP, for example, who can monitor both a user's "first-hop" link to Tor and the "last-hop" link from Tor to the user's communication partner, can correlate packets to de-anonymize flows [43]. Second, active disruption attacks can not only "deny service" but de-anonymize flows as well [6, 10]. Third, independent of the underlying anonymity protocols in use, widely-deployed tools often fail to isolate anonymous from non-anonymous communication state adequately, causing application-layer identity leaks via third-party browser plug-ins for example [1, 9, 17, 27].

As a step toward stronger anonymity and tracking protection we offer Dissent, a practical anonymous group group communication system resistant to traffic analysis. Dissent builds on and derives its strength from dining cryptographers or DC-nets [14, 36] and verifiable shuffles [11, 32, 44]. Prior systems to adopt these techniques, such as Herbivore [35, 49] and an earlier version of Dissent [20], demonstrated usability for anonymity sets only up to 40–50 participants, due to challenges in scaling and handling network dynamics. This paper improves the scalability of these strong anonymity techniques by at least two orders of magnitude, substantially narrowing the gap compared with relaying approaches [18, 21, 26, 38, 42].

Dissent derives scalability from an *anytrust* architecture [60]. A Dissent group consists of a potentially large set of *client* nodes representing users, and a smaller set of *servers*, facilitators of anonymous communication. Each client trusts that at least *any* one server will behave hon-

estly and not collude with the others against it, but the client *need not know or choose which server to trust*. While anytrust is not a new idea, Dissent rethinks DC-nets communication [14] around this model by sharing secret "coins" only between client/server pairs rather than between all node pairs, yielding a novel, practical and scalable system design. This design reduces clients' computation and communication burdens, and crucially in practical networks, decouples a group's overall communication performance from long "tail latencies" caused by slow, abruptly disconnected, or disruptive clients.

A Dissent prototype demonstrates usability on DeterLab with anonymity sets of over 5,000 members—over two orders of magnitude larger than anonymity sets demonstrated in comparable prior systems [20,35,49]. We expect Dissent to scale further with better optimization.

Although this paper's primary contribution is to show that strong anonymity can scale, Dissent also addresses certain disruption and information leakage vulnerabilities. In Tor and prior DC-nets schemes, an adversary who controls many nodes can anonymously disrupt partially-compromised circuits to increase the chance of *complete* compromise as circuits or groups re-form [10]. Dissent closes this vector with an *accusation* mechanism adapted to its anytrust network model, enabling a partially-compromised group to identify and expel disruptors without re-forming from scratch.

In local-area settings with low delay and ample bandwidth, Dissent can be used for anonymous interactive browsing with performance comparable to Tor. In this context Dissent can offer a strong local-area anonymity set complementing Tor's larger-scale but weaker anonymity. Dissent addresses an important class of anonymous browsing vulnerabilities, due to application-level information leaks [1, 9, 27], by confining the complete browser used for anonymous communication—including plug-ins, cookies, and other state—in a virtual machine (VM) that has no access to non-anonymous user state, and which has network access *only* via Dissent's anonymizing protocols.

Dissent has many limitations and does not yet address other weaknesses, such as long-term intersection attacks [39]. As a step toward stronger practical anonymity, however, this paper makes the following contributions:

1. An existence proof that traffic analysis resistant anonymity is feasible among thousands of participants.
2. A client/server design for DC-nets communication that tolerates slow or abruptly disconnecting clients.
3. A accusation mechanism offering disruption resistance in large-scale, low-latency DC-nets designs.
4. A VM-based browsing architecture enforcing a separation between anonymous and non-anonymous state.
5. Experiments demonstrating Dissent's usability in wide-area messaging applications, local-area interactive anonymity groups, and as a complement to Tor.

Section 2 of this paper describes Dissent's goals and how they relate to previous work. Section 3 presents Dissent's architecture. In Section 4, we overview our prototype, deployment models, and experiences. Section 5 presents the results of our experiments. We conclude with a summary of the paper's accomplishments.

2 Background and Related Work

This section outlines the state of the art in both practical anonymity systems and theoretical protocols, with a focus on the key security weaknesses that Dissent addresses.

2.1 Practical Anonymity on the Internet

Users can set "Do Not Track" flags [30] asking web sites not to track them. This advisory mechanisms asks the fox to guard the henhouse, however, relying on honest behavior from the web site and all network intermediaries. Even granted the force of law, such requests may be ignored by web sites in "grey markets" or foreign jurisdictions, just as today's anti-spam laws are ignored and circumvented.

For active protection against tracking or identification, centralized relay services such as Anonymizer [4] offer convenience but limited security, since one compromised server—or one subpoena—can break a user's anonymity. Users can create accounts under false names on popular services such as Facebook and Google+, but risk account loss due to Terms of Service violations—often for dubious reasons [48]—and may still be traceable by IP address.

For stronger protection without a single point of failure, decentralized relay networks [18, 21, 26, 38, 42] have proven practical and scalable. Relaying generally trades convenience against security, however, with some caveats [54]. Mixminion [21] forwards E-mail through a series of relays, delaying and batching messages at each hop to offer some traffic analysis protection. Tor [26], in contrast, consciously sacrifices traffic analysis protection to achieve low latencies for interactive Web browsing.

2.2 Anonymity Sets: Size versus Strength

The convenience that "weaker" systems such as Tor offer users may paradoxically give them a security *advantage* over "stronger" but less convenient systems such as Mixminion, because convenience attracts more users and thus yields much larger effective *anonymity sets* for their users to hide in. Tor only offers these large anonymity sets, however, *provided* the attacker is not capable of traffic

analysis—likely a reasonable assumption when Tor was designed. In today's more diverse global Internet, however, the adversary from whom users need identity protection may often be a national ISP controlled by an authoritarian state. Such an adversary realistically *can* monitor and "fingerprint" the traffic patterns of users and web sites *en masse*, completely de-anonymizing Tor flows that start and end within the same state. More recent traffic analysis attacks [6, 13, 37, 45] further accentuate this class of vulnerabilities. Thus, Tor may informally be viewed as offering a *potentially large but weak* anonymity set.

Two other approaches to anonymity theoretically offer security even against traffic analysis: verifiable shuffles [11, 32, 44], and "dining cryptographers" or DC-nets [14, 36, 58]. Communication and computation costs have in practice limited these methods to small anonymity sets, however. Herbivore [35, 49] supports mass *participation* by securely dividing large networks into smaller DC-nets groups, but guarantees each node *anonymity* only within its own group, showing scalability only to 40-node groups. The first version of Dissent [20] focused on accountability rather than scalability, combining verifiable shuffles with DC-nets to prevent anonymous disruption, but scaled only to 44-node groups. These techniques thus have so far offered *strong but small* anonymity sets.

Today's anonymity techniques thus present even well-informed users with a security conundrum: to use a tool like Tor that under favorable conditions hides them among tens of thousands of others, but under unfavorable conditions may not hide them at all; or to use a tool that can offer only a small anonymity set but with higher confidence. Dissent's goal is to alleviate this conundrum.

3 Dissent Architecture

This section first summarizes DC-nets, then details how Dissent achieves scalability and resilience to slow or unreliable clients. It finally outlines how Dissent traces disruptors and schedules rounds, and current limitations.

3.1 DC-nets Overview and Challenges

In classic DC-nets [14], one anonymous sender in a group wishes to share a message with fellow group members. To exchange a 1-bit message, every member shares a secret random coin with each of the other $N-1$ members. Every pair together first flips their shared coin, agreeing on the outcome. Then each member individually XORs together the values of all the coins he shares, while the anonymous sender additionally XORs in his 1-bit message, to produce the member's *ciphertext*. Finally, all members broadcast their ciphertexts to each other. Since each coin is XORed into exactly two members' ciphertexts, all shared coins cancel, revealing the anonymous sender's message without revealing who sent it. For longer messages, the group uses multiple coin flips—in practice, cryptographic pseudo-random number generators (PRNGs) seeded by pairwise shared secrets.

Practical implementations of this conceptually simple design face four key challenges: scheduling, disruption, scalability, and network churn. First, since DC-nets yield an Ethernet-like broadcast channel, in which only one member can transmit anonymously in each bit-time without colliding and yielding garbled output, an *arbitration* or *scheduling* mechanism is needed. Second, any misbehaving member can anonymously disrupt or "jam" the channel simply by transmitting random bits all the time. Dissent builds on and extends several prior approaches to address these challenges [20, 36, 58].

The third challenge directly limits scalability. Every member normally shares coins (or keys) with every other member, so each node must compute and combine $O(N)$ coins for every bit of shared channel bandwidth. Computing ciphertexts via modular arithmetic instead of XORed bits [36] can address this issue asymptotically, but at a large constant-factor cost. Communication cost can also limit scalability if every node broadcasts its ciphertext to every other. In Herbivore [35, 49] a single node collects and combines ciphertexts for efficiency, but this leader-centric design offers no reliable way to identify anonymous disruptors without re-forming the group, leaving groups vulnerable to DoS attacks against anonymity [10].

The fourth challenge, network churn, indirectly limits scalability in practice. As each member shares a coin with every other, a round's output is indecipherable until *all* members submit their ciphertexts. Thus, one slow member delays the entire group's progress. If any member disconnects during a round, all other members must recompute and rebroadcast their ciphertexts anew. Beyond "normal-case" churn, an adversary who controls f group members can take them offline one one at a time to force a communication round to timeout and restart f times in succession. Threshold cryptography can address this issue in non-interactive scenarios [36], but may be too heavyweight for interactive communication.

3.2 Design and Deployment Assumptions

Dissent assumes a cloud-like *multi-provider* deployment model illustrated in Figure 1, similar to the model assumed in COR [38]. A Dissent *group* consists of a possibly large number of *client* nodes representing individual users desiring anonymity within the group, supported by a small number of reliable and well-provisioned cloud of *servers*. We assume each server is run by a respected, technically competent, and administratively independent

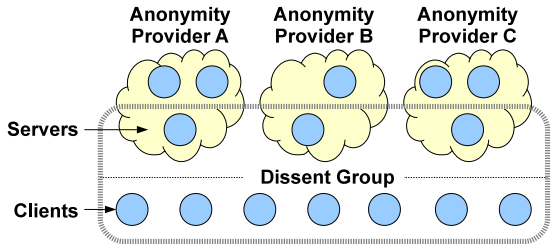

Figure 1: Dissent's multi-provider *anytrust* cloud model

anonymity service provider. We envision several commercial or non-profit organizations each deploying a cluster of Dissent servers to support groups, as either a for-profit or donation-funded community service.

For anonymity and other security properties Dissent relies on an *anytrust* assumption [60]. Clients need not rely on *all* or even *particular* providers or their servers being honest. Instead, each client trusts only that *there exists* at least one provider—*any* provider—who is honest, technically competent, and uncompromised. Clients *need not know or guess which provider's server is the most trustworthy.* Later sections detail how Dissent's design relies on this assumption to achieve scalability.

This paper focuses on the operation of a single group—which forms an anonymity set from a user's perspective—and leaves out of scope most details of how groups are formed or subsequently administered, how providers deploy their services or scale to support many groups, etc. As a simple group formation mechanism we have prototyped, an individual creates a file containing a list of public keys—one for each server (provider) and one for each client (group member)—then distributes this *group definition* file to the clients and servers. A cryptographic hash of this group definition file thereafter serves as a *self-certifying identifier* for the group [31], avoiding membership consensus and PKI issues at the cost of making the group's composition static. The group formation techniques explored in Herbivore [35,49] could offer complementary methods of forming Dissent groups dynamically.

3.3 Dissent Protocol Outline

To initiate communication, a group's servers periodically run a scheduling process described later in Section 3.10. This process yields a list of *pseudonym keypairs*, one for each participating client. All nodes know and agree on the list of public keys and their order, and each client knows the private key for its slot in the DC-net defined by the ordered list of pseudonym keypairs, but neither clients nor servers know which clients hold which *other* slots. This list schedules subsequent DC-nets rounds as shown in Figure 2, and enables the protocol to offer accountability as described later in Section 3.9.

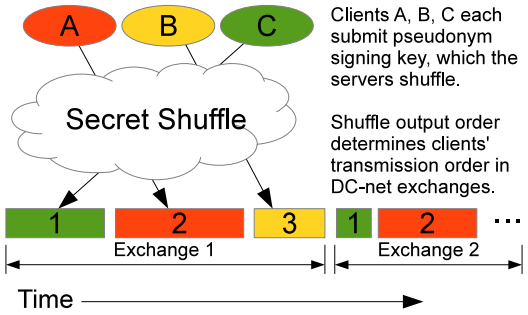

Figure 2: DC-nets scheduling via a verifiable shuffle

After setup, group members commence a continuous series of rounds. Each round allows the owner of each slot to transmit one or more bits anonymously, as defined by the schedule and information from prior rounds. During a round, each client first generates M pseudo-random strings, each based on a secret key he shares with each of the M servers, and XORs these strings together. To send a message, the client additionally XORs his cleartext message into the bit positions corresponding to his anonymous transmission slot. The client then transmits his ciphertexts to one or more servers, then waits.

The servers collect as many client ciphertexts as possible within a time window. At the end of this window, the servers exchange with each other the list of clients whose ciphertexts they have received. Each server then computes one pseudo-random string for each client that submitted a ciphertext, using the secret shared with that client. The server XORs these strings, together with with the client ciphertexts the server received, to form the server's ciphertext (Figure 3). The servers then distribute their ciphertexts among themselves. Upon collecting all server ciphertexts, each server XORs them to reveal the clients' cleartexts, and distributes the cleartexts to the clients connected to them, completing one DC-net round. Successive DC-net rounds ensue.

The rest of this section describes Dissent's client and server protocols, summarized in Algorithms 1 and 2, respectively, and in Figure 4. All network messages are signed to ensure integrity and accountability, but we omit these signatures to simplify presentation.

3.4 Secret Sharing in the Anytrust Model

The key to Dissent's scalability and resilience to churn is its client/server secret-sharing graph. Unlike the "all-to-all" secret-sharing graph in most DC-nets designs, Dissent shares secrets *only* between all client/server pairs.

As formalized by Chaum [14], the anonymity set an honest node obtains via DC-nets consists of the node's connected component in the secret-sharing graph, after

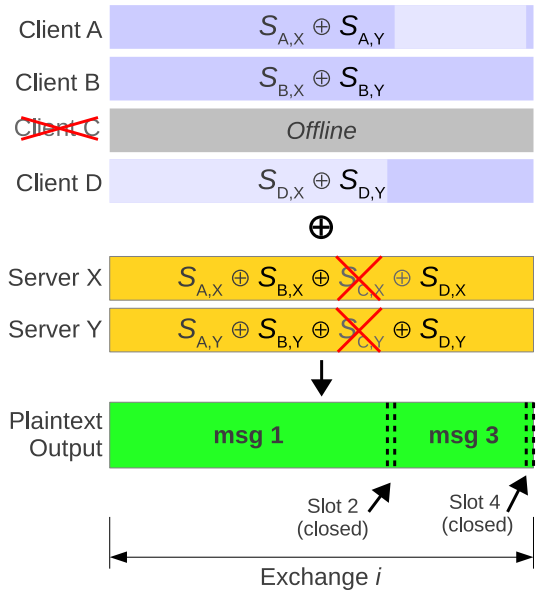

Figure 3: Dissent round structure. Each client-server pair shares a secret pseudo-random string $s_{i,j}$. Client D anonymously transmits a message in slot 1 and client A anonymously transmits a message in slot 3. Servers do not XOR in the strings for offline client C.

removing dishonest nodes and their incident edges from the graph. A sparser secret-sharing graph thus reduces a node's anonymity, compared with the ideal anonymity set consisting of all honest nodes, if and only if the dishonest nodes partition the honest nodes into multiple connected components. Because each Dissent client shares a secret with each server, the honest nodes remain connected—yielding an ideal anonymity set—if and only if there is at least one honest server. This is precisely what Dissent's anytrust model assumes. The downside is that if *all* servers maliciously collude, clients obtain *no* anonymity.

A direct benefit of Dissent's client/server secret-sharing is that clients enjoy a lighter computational load during exchanges. Each client shares secrets with only the $M \ll N$ servers, thus clients need only compute M pseudo-random bits for each effective bit of DC-net channel bandwidth. Each of the servers must compute N pseudo-random bits per cleartext bit, as in traditional DC-nets, but these computations are parallelizable, and Dissent assumes that the servers are provisioned with enough computing capacity to handle this load. Just as important in practice, however, are the model's indirect benefits to network communication and resiliency, detailed next.

3.5 Optimizing Network Communication

Dissent leverages its client/server architecture to reduce network communication overhead. In conventional DC-nets, all nodes broadcast messages to all other nodes. Dissent reduces the number of communication channels by a factor of N by organizing clients and servers into a two-level hierarchy. Clients communicate with only a single server, and servers communicate with all other servers.

Algorithm 1 Dissent Client DC-net Protocol

1. **Scheduling:** Each client i creates a fresh secret *pseudonym key*, $k_{\pi(i)}$, then encrypts and submits it to a key-shuffle protocol, which permutes and decrypts all clients' keys, giving client i a secret *permutation slot* $\pi(i)$ unknown to all other nodes. A well-known scheduling function $S(r, \pi(i), H)$ determines the set of bit-positions client i owns in each subsequent DC-nets round r, after a history H of prior round outputs.
2. **Submission:** Each client i forms a cleartext message m_i containing arbitrary data in the bit-positions i owns according to $S(r, \pi(i), H)$, and zero elsewhere. From secrets K_{ij} that client i shares with each server j, i computes pseudo-random strings $c_{ij} = \text{PRNG}(K_{ij})$. Client i then XORs these strings with message m_i to produce ciphertext $c_i = m \oplus c_{i1} \oplus \cdots \oplus c_{iM}$, which i signs and transmits to one or more servers.
3. **Output:** Each client i waits for a message from any server containing round r's cleartext output signed by all servers: $(r, \vec{m}, \vec{sig})$. Client i verifies all servers' signatures, extracts the messages in all slots, then proceeds to round $r + 1$ by repeating from step 2.

This optimization does *not* make a client's anonymity dependent on the particular server it chooses to connect to, because anonymity depends on the secret-sharing graph described above and not on physical communication topology. Since each client shares a secret with *all* servers, even if a client's directly upstream server is malicious, that server cannot decode the client's anonymous transmissions except with the cooperation of all the servers, including the honest one we assume exists. A server can DoS-attack an attached client by persistently dropping its submitted ciphertexts, but the client will recognize such an attack from the absence of its cleartexts in the group's output—which *all* servers must sign—signaling the client to switch to a different server.

To reduce communication costs further, servers locally combine their pseudo-random strings with their clients' ciphertexts. Servers thus avoid forwarding individual client ciphertexts to other servers, reducing total communication complexity from $O(N^2)$ to $O(N + M^2)$ when $M \ll N$. Related optimizations reduce the number of signature verifications a client performs from $O(N)$

Algorithm 2 Dissent Server DC-net Protocol

1. **Submission:** In each round r, each server j collects ciphertexts c_i from some clients i, until all of j's directly-connected clients have responded or the round closure deadline has passed.
2. **Inventory:** Server j forms a list $\vec{l_j}$ of client identities from whom j has received ciphertexts by the deadline, then broadcasts this list to the other servers.
3. **Commitment:** Given all servers' vectors l_j, the servers deterministically trim redundant entries for clients who submitted ciphertexts to multiple servers, yielding new lists l'_j, then form a composite client list $l = \bigcup_j l'_j$. If the round r participation count $p_r = |l|$ is below a policy-defined fraction α of the previous round's participation count p_{r-1}, the servers return to step 1 and wait for more clients to submit ciphertexts.

   Otherwise each server j computes pseudo-random strings $s_{ij} = \text{PRNG}(K_{ij})$ from the shared secrets K_{ij} of clients $i \in l$, and XORs these strings with the client ciphertexts j received directly, forming server ciphertext $s_j = (\bigoplus_{i \in l} s_{ij}) \oplus (\bigoplus_{i \in l'_j} c_{ij})$. Server j computes a commit $C_j = \text{HASH}(s_j)$ and sends C_j to all servers.
4. **Combining:** Upon receiving all other servers' commit, server j shares s_j with the other servers.
5. **Certification:** The servers verify $C_j = \text{HASH}(s_j)$ for all j, and form cleartext output $\vec{m} = \bigoplus_j s_j$. Each server j signs $\vec{m}$, and sends its sig_j to all servers.
6. **Output:** Servers collect all signatures sig_j into $\vec{sig}$, then distribute $(r, \vec{m}, \vec{sig})$ to directly attached clients.

to $O(M)$ and the number of messages clients must parse from $O(N)$ to $O(1)$ (see Algorithm 1, steps 2, and 3).

3.6 Tolerating Network Churn

More important than reducing the computational load on clients, Dissent's client/server secret-sharing graph enables the servers to collaborate to reduce the group's vulnerability to client disconnection and churn, as well as deliberate DoS attacks by malicious clients. In conventional online DC-nets, if any member goes offline, all other members must recompute and resend new ciphertexts, omitting the PRNG stream they shared with the failed member. The chance that a given round will have to be "re-run" in this way increases dramatically as group size and client churn increase.

Since Dissent clients share secrets only with the servers and not with other clients, a client's ciphertext is independent of the online status of other clients. The servers can therefore complete a messaging round even if some clients

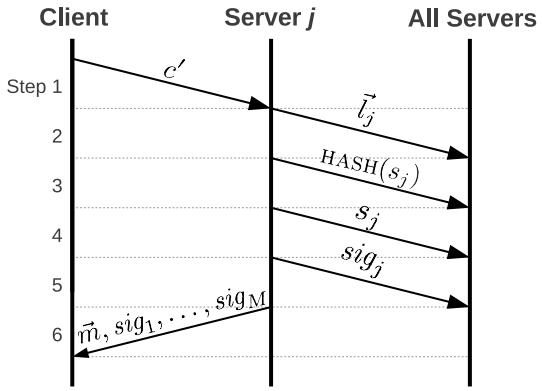

Figure 4: Dissent DC-net protocol.

disconnect at arbitrary points during the round, or otherwise fail to deliver their ciphertexts before a deadline. The servers first collect those client ciphertexts that arrive in time, then agree among each other on the complete set of client ciphertexts available (the union of all servers' client ciphertext sets), and finally XOR these client ciphertexts with the pseudo-random strings each server shares with those clients to form the servers' ciphertexts. Thus, client delays or disconnections never require servers to interact with clients iteratively within the same round, as they would in standard DC-nets to obtain revised ciphertexts.

3.7 Participation and Anonymity Metrics

To ensure "strength in numbers," users may wish to send anonymous messages only when at least some number of other group members are online and participating. Since the servers know the set of clients who are online and successfully deliver ciphertexts each round, the servers publish a *participation count* for each round. A user who judges this count to be too low can continue to participate passively in the group but send only an empty ("null") message in each round until participation increases.

Servers can publish participation counts only for *past* rounds, but clients can come and go at any time. A client thus might decide to send a message on the basis of one round's high participation count, and submit a sensitive message in the next round, only to discover after the round completes that far fewer clients remained online or delivered their ciphertexts in time. A powerful adversary might even start a DoS attack against many honest clients just as a sensitive anonymous posting is anticipated, in hopes of isolating and exposing the poster this way.

To address such risks, if the last round's participation count was P, the servers will not complete the next round until at least αP clients submit ciphertexts, where $0 \leq \alpha \leq 1$ is a policy constant defined at group creation time. If fewer than αP clients submit ciphertexts by the round's

deadline, the servers keep waiting until at least αP clients show up, or until a much longer *hard timeout* occurs. On a hard timeout, the servers discard all clients' ciphertexts, report the round as failed, and publish a new participation count on whose basis the clients make fresh decisions for the next round. The fraction α thus limits the rate at which participation may decrease unexpectedly round-to-round.

While Dissent can guarantee users that a sensitive message will be posted only when participation is at some threshold level, participation count unfortunately offers only an estimate of anonymity set size. If some participants are dishonestly colluding with the adversary, a client is anonymous only among the set of *honest* participants. A group's risk of infiltration of course depends on how it is formed and managed. In our current approach where a group is defined by a static list of public keys (Section 3.2), the dishonest members are those whose public keys the adversary manages to persuade the group creator to include in the list at definition time, plus any formerly-honest members who the adversary might compromise after group formation. Since users cannot ultimately know how many of their peers may be "spies," estimating anonymity necessarily remains subjective.

3.8 Eliminating Empty Slot Overhead

In typical blogging or chat applications we expect many clients to be silent much of the time, sending null messages in most rounds and real messages only occasionally. To optimize this common case, Dissent's scheduling scheme gives each client two slots: a one-bit *request* slot and a variable-length *message* slot. Initially the message slot is *closed*, with length 0. When a client sets its request bit in round r, its message slot *opens* to a fixed size in round $r+1$. The message slot includes a length field, with which the client can adjust the slot's length in subsequent rounds: to send a larger message efficiently in round $r+2$, for example, or to close its message slot back to length 0.

A dishonest member could DoS attack another client by guessing when the victim will transmit and sending a 1 in the victim's request slot, cancelling the victim's open request. To address such attacks, a client first sets its request bit unconditionally, but if its slot fails to open, the client randomizes its request bit in subsequent rounds, ensuring success after $t+1$ rounds with probability $1-(\frac{1}{2})^t$.

3.9 The Accusation Process

Dishonest members can disrupt DC-nets in general by XORing non-zero bits into other members' slots, corrupting the victim's cleartext. Earlier approaches to this challenge used complex *trap* protocols [58], expensive pairing-based cryptography [36], or required a costly shuffle before every DC-nets round [20]. Herbivore [35] mitigates the risk of disruption by limiting clique size and the rate at which disruptors can join them, at the cost of small anonymity sets and potentially increased vulnerability when disruptors are common [10].

Dissent introduces a new accusation scheme that adds little overhead in the absence of disruptors, but enables the servers to identify and expel a persistent disruptor quickly with high probability. The overall scheme operates in three stages. First, the victim of a disruption must find a *witness bit* in some round's DC-net output, which we define as a bit that was 0 in the victim's cleartext, but which the disruptor flipped to a 1. Second, the victim anonymously broadcasts an *accusation*, a message signed with the victim's pseudonym key identifying the witness bit. Finally, the servers publish all PRNG outputs that contributed to the client and server ciphertexts at the witness bit position, using them to trace the client or server that XORed an unmatched 1 bit into this position. The signed accusation attests that the traced node must be a disruptor.

The first challenge is ensuring that a disruption victim can find a witness bit. If a disruptor could predict the victim's cleartext output—or discover it from other honest nodes' ciphertexts before computing its own ciphertext—then the disruptor could avoid leaving witness bits by flipping only 1 bits to 0 in the victim's slot. To make all cleartext bits unpredictable, clients apply a cryptographic padding scheme analogous to OAEP [7]. The client picks a random seed r, generates a one-time pad $s = \text{PRNG}\{r\}$, XORs it with the original message m, and transmits $r||m \oplus s$ in the client's message slot. Since clients submit their ciphertexts before the servers compute theirs, and the commitment phase in Algorithm 2 prevents dishonest servers from learning honest servers' ciphertexts before computing their own, any disruptive bit-flip has a $1/2$ chance of producing a witness bit.

The second challenge is enabling the victim to transmit its accusation: if it did so via DC-nets, the disruptor could simply corrupt that transmission as well. To avoid this catch-22, Dissent falls back on the less efficient but disruption-resistant verifiable shuffle it uses for scheduling. Each client's DC-net message slot includes a k-bit *shuffle request* field, which the client normally sets to 0. When a disruption victim identifies a witness bit, it sets its shuffle request field in subsequent rounds to a k-bit random value. Any nonzero value signals the servers to start an accusation shuffle, in which the victim transmits its signed accusation. The disruptor may try to squash the shuffle request, but succeeds with at most $1-(\frac{1}{2})^k$ chance, and the victim simply retries until it succeeds.

The final challenge is tracing the actual disruptor. An accusation consists of the round number in which the dis-

ruption occurred, a slot index $\pi(i)$, and the index k of a bit the disruptor flipped from 0 to 1 in this slot, all signed by the slot owner's pseudonym key, $k'_{\pi(i)}$. On receiving an accusation, the servers verify its signature, and check that the indicated bit was indeed output as 1. The servers then recompute and exchange all the individual PRNG bits that the clients and servers should have XORed together to compute their ciphertexts: $c_{ij}[k]$ in Algorithm 1 and $s_{ij}[k]$ in Algorithm 2. Each server independently attempts to find a mismatch, where: (a) a server did not transmit the full set of client ciphertext bits; (b) the accumulation of transmitted bits do not match what the server sent out earlier: $s_j[k] \neq (\bigoplus_{i \in l} s_{ij}[k]) \oplus (\bigoplus_{i \in l'_j} c_i[k])$; or (c) the client's ciphertext bit does not match the accumulation across servers: $\bigoplus_j s_{ij}[k] \neq c_i[k]$, for some client i. The first two cases trivially expose a server as dishonest. In the final case, each server requests from client i a rebuttal on why the set of server bits, $s_{ij}[k]$, are incorrect, namely which server equivocated. An honest client can respond with the malicious server's identity, their shared secret, and proof of this shared secret.

3.10 Scheduling via Verifiable Shuffles

Dissent uses verifiable shuffles [11, 32, 44] both to schedule and distribute pseudonym keys for subsequent DC-nets rounds, and for transmitting accusations to servers. Clients submit messages (or keys to be anonymized) to the shuffle protocol, and the shuffle outputs a random permutation of these messages (or keys), such that no subset of clients or servers knows the permutation. Dissent depends minimally on the shuffle's implementation details, so many shuffle algorithms should be usable.

Dissent uses Neff's verifiable shuffle [44] to create verifiable secret permutations, and Chaum-Pedersen proofs [15] for verifiable decryptions. To shuffle, each client submits an ElGamal-encrypted group element. In a general message shuffle, clients embed their messages within a group element, encrypt it with a combination of all server keys, then transmit it to first server, who shuffles the input and removes a layer of encryption. Each server shuffles and decrypts in turn, until the last server reveals the cleartexts and distributes them to all clients.

The design supports both general message shuffles and more efficient *key shuffles*. Since Neff's algorithm shuffles ElGamal ciphertexts, general messages must be embedded within group elements. The entries of a key shuffle are already group elements, however, thus requiring no message embedding. Key shuffles also permit the use of more computationally efficient groups that are suitable for keys but not for message embedding.

3.11 Limitations

This section discusses a few of Dissent's shortcomings and possible ways to address them in future work.

Large networks with many groups Our evaluations (Section 5) demonstrate that a single Dissent network can accommodate over 5,000 clients. To be broadly usable at Internet scale, Dissent must scale to much larger network sizes, of hundreds of thousands of nodes or more. One way to serve very large networks would be to adopt a technique introduced by Herbivore [35]: break the overall network into smaller parallel Dissent groups—with tens of servers and thousands of clients each. A secure join protocol, as in Herbivore, could protect a single session from being overrun with Sybil identities.

Intersection attacks Dissent's traffic analysis resistance does not protect against *membership intersection attacks*, where an adversary correlates linkable anonymous transmissions to changes in clients' online status. If an anonymous blogger posts a series of messages, each signed by the same pseudonym but posted in different rounds, and the adversary sees that only Alice was online in *all* of those rounds (though many other group members were present in *some* rounds), the adversary might pinpoint Alice as the blogger. There is no perfect defense against intersection attacks when online status changes over time [39]. Dissent users could gain some protection against the intersection attack by avoiding linkable anonymous transmissions (e.g., the use of pseudonyms). Alternatively, users could adopt a "buddy system," transmitting linkable cleartexts only when *all* of a fixed set of "buddies" are also online. With certain caveats, this discipline ensures that a user's anonymity set includes at least his honest buddies, at the availability cost of making the user unable to transmit (safely) when *any* buddy is offline.

Handling server failure Dissent addresses network churn only among clients: if a server goes offline, the protocol halts completely until all servers are available again or the group is administratively re-formed to exclude the failed server (which currently amounts to creating a new group). We expect that Byzantine fault-tolerance techniques [12] could be adapted to mask benign or malicious server failures, at a cost of imposing a stronger security assumption on the servers. In a BFT group designed to tolerate f concurrent failures, for example, client anonymity would likely depend on at least $f+1$ servers being honest, rather than just one. A malicious group leader could form a live "view" deliberately excluding up to f honest and online servers, replacing them with f dishonest servers who appear live and well-behaved but privately collude in attempt to de-anonymize clients.

Group management and server selection As discussed in Section 3.2, Dissent groups currently contains a static list of clients and servers; allowing more dynamic group administration while maintaining security remains an important challenge. In a public Dissent deployment with hundreds of servers and thousands of parallel groups, users would benefit especially from automatic server selection. Since the user must trust at least one of the servers, a server selection algorithm might have to consider which servers user trusts, how close the user is to which servers, a server's reliability, and other security and performance factors. Dissent's server selection problem is likely analogous to the path selection problem in Tor [6, 28], and might build on prior work on this topic.

Mobile devices In the United States, consumers now use mobile phones more for Internet browsing and non-voice data transfer than for making phone calls [61]. As everyday computing shifts to mobile devices, an ongoing challenge is to offer users the same privacy protections on phones as they would have on desktop computers [8,33,34]. We have yet to deploy or test Dissent on mobile devices, but expect Dissent's computation and communication optimizations to be useful in this context.

Formal security analysis While Dissent is based on techniques with formal security proofs [14, 15, 44], a full formal analysis of Dissent remains for future work.

4 Implementation

This section describes the current Dissent prototype and how we have applied it to two anonymous communication use cases: wide-area group messaging and local-area anonymous Web browsing.

4.1 Prototype Overview

We have implemented Dissent in C++ with the Qt framework and the CryptoPP cryptography library. The prototype implements the complete Dissent anonymity protocol along with the accountability sub-protocols described in Section 3.9. The system assumes the existence of a certificate authority (or other entity) that manages the long-term public keys of all servers and clients. The prototype also assumes that participants have used an outside channel to agree upon a common set of servers. Source code may be found at the Dissent project home page.[1]

User applications interact with a node running our Dissent prototype using HTTP API calls or a SOCKS proxy interface. The HTTP API allows clients to post raw messages (byte-strings) directly into the protocol session. The prototype's SOCKS v5 proxy allows users to tunnel TCP and UDP traffic flows transparently through the Dissent protocol session. One or more nodes in the Dissent network serve as SOCKS entry nodes, which listen for incoming SOCKS proxy requests from user applications (e.g., Skype or Firefox). The entry node accepts SOCKSified traffic flow from the user application, assigns the flow a random identifier (to allow a receiving node to distinguish between many flows), adds destination IP and port headers to the flow, and sends it into the active Dissent protocol round. A single SOCKS exit node (who is a non-anonymous protocol participant) reads the tunneled traffic from the Dissent protocol round, forwards it over the public network to the destination server, and sends the response back through the Dissent session.

[1] `http://dedis.cs.yale.edu/2010/anon/`

4.2 Anonymous Microblogging Application

Dissent's decentralized architecture and trust model make it potentially attractive as a substitute for commercial microblogs in high-risk anonymous communication scenarios. Since Dissent relies on no single trusted party, we expect it to be much more challenging even for a powerful adversary—such as an authoritarian government or its state-controlled ISP—to identify an anonymous blogger without compromising *all* participating Dissent servers.

In our evaluation section, we present performance results for a prototype microblogging system running on PlanetLab with up to 2,000 nodes and DeterLab with up to 5,000 nodes. A simple chat-like Web interface allows users to post short messages into an Dissent protocol session using our HTTP API. Our results suggest Dissent could form a practical platform for Internet-scale microblogging in situations requiring stronger security properties than the current commercial platforms offer.

4.3 Local-Area Web Browsing

When deployed in a local-area network, Dissent can provide interactive communication with local-area anonymity: requests are anonymous among a local set of users. To demonstrate this use of Dissent, we have developed *WiNoN*, a system that uses virtual machines to isolate a user's identifiable OS environment from their anonymous browsing environment, an important issue given that browser signatures have a reasonable chance of uniquely identifying a user [27].

In WiNoN, depicted in Figure 5, the Dissent client software runs on the host OS with network traffic from the WiNoN VM tunneled through the Dissent SOCKS proxy. Since applications in the WiNoN VM have no access to the network interface or to the user's non-anonymous storage, they are unable to learn the user's long-term identity (unless the user inadvertently enters some identifiable information into the WiNoN VM).

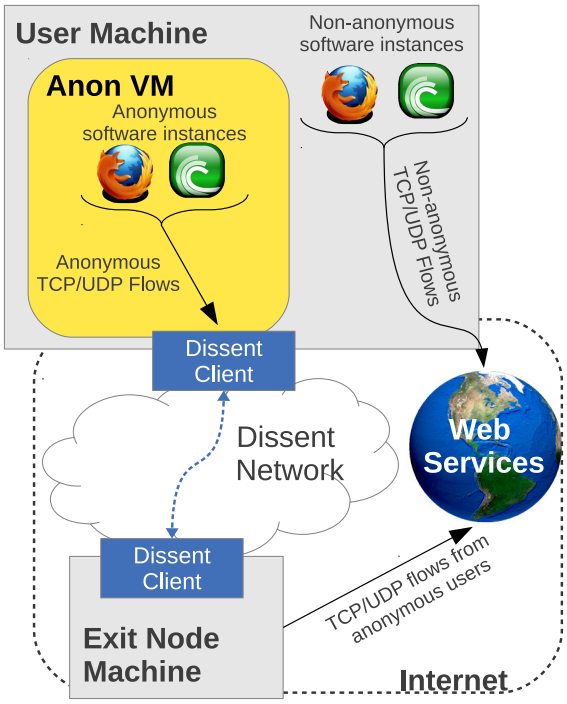

Figure 5: WiNoN system diagram. Traffic from the anonymous VM flows through an Dissent tunnel to an exit node. The exit node reads traffic from the tunnel and forwards it onto the Internet on behalf of the WiNoN client.

On simulated WiFi networks with tens of nodes and typical bandwidth and delay parameters, we find the WiNoN anonymization network fast enough for browsing the Internet and streaming videos. Prior work uses virtual machines to isolate network environments [41] and tunnel traffic through Tor [55]. No previous system to our knowledge, however, enables a user to run Flash movies, Skype, and other untrusted applications safely and anonymously.

5 Evaluation

This section first examines Dissent's ability to handle unreliable client nodes. Next we evaluate Dissent's performance and scalability in instant-messaging and data sharing scenarios with varying numbers of clients and servers. We then explore the costs of different protocol stages: the initial key shuffle, a DC-net exchange, and finally the accusation process. Finally, we evaluate the performance of Dissent applied to the WiNoN Internet browsing application described in section 4.3.

In our evaluations, we used the Emulab [29], DeterLab [23], and PlanetLab [16] testbeds, and nodes from Amazon's EC2 service. Emulab and DeterLab offer controlled, repeatable conditions on isolated networks, while

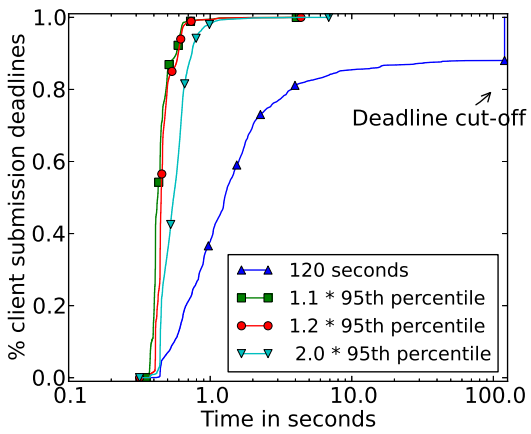

Figure 6: A CDF plot demonstrating the time for a message exchange to complete when using four message window policies.

we used PlanetLab nodes on the public Internet to offer a more realistic test of Dissent's ability to handle client delays and churn. For our evaluations on the public Internet, we used eight server machines—one located at our university and seven at EC2 sites—located at unique locations on four different continents, and we used the entire set of available PlanetLab nodes as clients. We indicate the exact number of PlanetLab clients where applicable.

5.1 Slow and Unreliable Clients

On public networks, distributed systems must cope with slow and unreliable machines [47]. Dissent's servers prevent slow nodes from impeding the protocol's overall progress by imposing a ciphertext submission window. Once the client submission time window has closed, servers continue executing the protocol even if every client has not submitted a ciphertext. Larger windows potentially allow more clients to participate in each message exchange, but increase messaging latency. Smaller windows size reduce latency of exchanges but might prevent slower clients from participating.

To help us select an effective window closure policy for our evaluations on PlanetLab, we collected a data trace from a Dissent deployment with over 500 clients running on PlanetLab nodes and eight servers running on EC2, using a static window size of 120 seconds. The exact number of clients varied over the course of the 24-hour evaluation period. We used the data from this PlanetLab experiment to test a variety of window closure policies.

To ensure that most clients are able to participate in each message exchange, we do not close the submission window until at least 95% have submitted messages. Once

95% of clients submit messages, we multiply the time elapsed by a constant factor to determine window time.

The fraction of clients who missed the submission window decreased as this multiplicative constant increased: $1.1\times$: 2.3%, $1.2\times$: 1.5%, and $2\times$: 0.5%. The data in Figure 6 demonstrate that the client submission time is not very sensitive to multiplicative constant used. For the rest of the evaluation, we chose the $1.1\times$ policy, since there was not significant variation among the three.

Regardless of the specific window closure policy chosen, Figure 6 demonstrates the importance of insulating the group's progress from that of its slowest clients in an unpredictable environment like PlanetLab. In the baseline case where the servers wait until *all* clients submit or a 120-second hard deadline is reached, 50% of DC-net rounds are delayed by "stragglers" by an order of magnitude or more compared with early-cutoff policies, and 15% of rounds are delayed until the 120-second hard deadline versus almost none with early cutoff policies.

5.2 Wide-Area Applications

To evaluate Dissent's usability in wide-area microblogging or data sharing scenarios, as described in Section 4.2, we evaluated the protocol on both DeterLab [23] and PlanetLab [16]. On DeterLab, which offered controlled test conditions and greater hardware resources, we evaluated both a microblog and data sharing like behavior; we evaluated only the microblog scenario on PlanetLab.

To simulate a plausible traffic load in the microblog scenario, a random 1% of all clients submit 128-byte messages during any particular round. In the data sharing scenario, one client transmits a 128KB message per round.

Due to time and resource limitations, the DeterLab evaluation used two system topologies: 32 servers with 10 client machines per server, and 24 servers with 12 client machines per server, for a total of 320 and 288 client machines respectively. To simulate a larger number of Dissent client participants than we had physical testbed machines for, we ran up to 16 Dissent client processes on each client machine, for up to 5120 client processes.

In the testbed topology, servers shared a common 100 Mbps network with 10 ms latency, while clients shared a 100 Mbps uplink with 50 ms latency to their common server. We intended the servers to share a 1000 Mbps network, but the testbed did not support this configuration.

For the PlanetLab tests, we deployed 17 servers: 16 on EC2 using US East servers and one control server located at Yale University. The latency between Yale and EC2 was approximately 14 ms round trip. This clustered setup is intended to represent a deployment scenario in which multiple organizations offer independently-managed Dissent servers physically co-located in the same or geographically nearby data centers, facilitating high-bandwidth and low-latency communication among the servers while keeping their management decentralized for security.

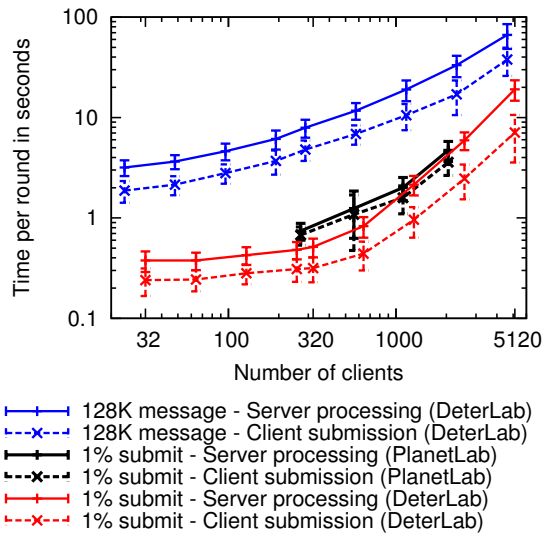

Figure 7: Time per round in microblog (1% submit) and data sharing scenarios, for varying number of clients.

Figure 7 shows the system's scalability with client load by varying the number of clients relying on a static set of 32 servers. Figure 8 in turn varies the number of servers while maintaining a static set of 640 clients. At smaller group sizes, additional servers do not benefit performance. As demands on the servers scale, however, their utility becomes more apparent, especially in the 128K message scenario. Performance appears to be dominated by client delays, namely the time between clients receiving the previous round's cleartext and the servers receiving the current round's ciphertext message. However, in comparing the PlanetLab evaluation to the DeterLab evaluation and server size of 1, we can ascertain that latency between servers tends to dominate delays in that environment, though computational load is not negligible.

The prototype shows greatest usability for group sizes up to 1,000; thereafter delays become longer than 1 second in the microblogging scenario. At best, delays were on the order of 500 to 600 ms for 32 to 256 clients. In the static client network, varying server count, showed time increases on server-related aspects of the protocol but reduced time on client-related aspects. We therefore expect that with greater demand—either in terms of nodes or bandwidth—client-related costs are likely to dominate.

In comparing the microblogging and 128K message scenarios, the graphs suggest that bandwidth tends to dominate for larger messages and latency for smaller messages. Most importantly, the evaluations suggest that

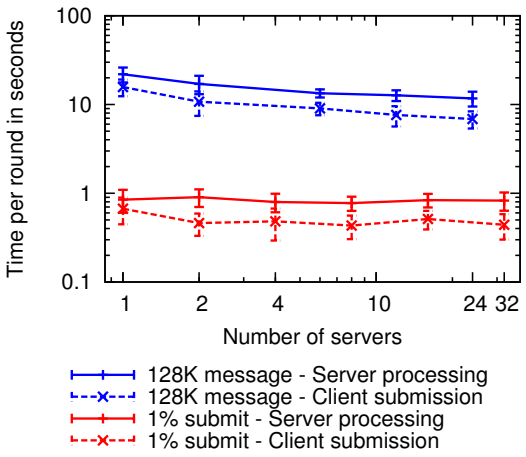

Figure 8: Time per round in microblog (1% submit) and data sharing scenarios, for varying number of servers supporting 640 clients.

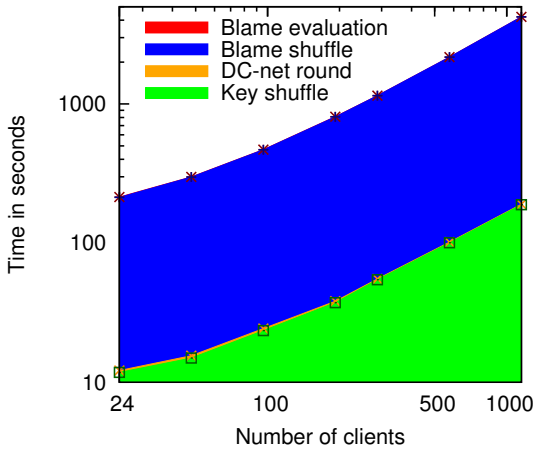

Figure 9: Time elapsed during a whole protocol run for varying client sizes, 24 servers, and 128 byte messages.

Dissent can support delay-sensitive applications like microblogs and instant messaging.

5.3 Full System Evaluation

While the previous experiments focused on measuring DC-nets rounds, the primary focus of this paper, we now explore time durations in a single full execution of the entire Dissent protocol: key shuffle, a single DC-net exchange, accusation shuffle, and accusation tracing. Our results shown in Figure 9 used the same DeterLab configuration consisting of 24 servers with 12 clients each configuration as described in the previous section. In contrast to verifiable mix-nets, the Dissent protocol's DC-nets round is extremely efficient, accounting for a negligible portion of total time in large groups.

The time difference between accusation and key shuffles illustrate the performance benefits of the key shuffle discussed in Section 3.10. In small groups the accusation shuffle is reasonably fast, but in larger groups its cost increases quickly, to over an hour for 1,000-client groups.

5.4 Web Browsing: Dissent and Tor

To explore the practicality of Dissent for local-area anonymity as described in section 4.3, we deployed a smaller-scale Dissent network of 5 servers and 24 clients on the Emulab [29] network testbed. The testbed's experimental network topology approximated the characteristics of a small WiFi network: each node was connected to a central switch via a 24 Mbps link with 10 ms of latency. One of the servers acted as a gateway connecting the private test network to the public Internet.

In this environment, we ran an automated HTTP browser on one of the client nodes to download the index pages from each site on the Alexa "Top 100" Web sites [3] from the real public Web server. For each index page, the client requested the HTML page and then recursively and concurrently requested dependent assets (images, CSS, JS, etc.). Although we used an automated HTTP browser for these trials, Dissent supports standard Web browsers as well.

We used four different network configurations to test Dissent's performance under four deployment scenarios. In the first scenario, no anonymity, the gateway connects directly to the public Internet. The second scenario, Tor alone, shows the performance of state-of-the-art wide-area anonymous Internet access. We emphasize that we compare with Tor only to provide a general reference point for gauging Dissent's usability: this is by no means an "apples-to-apples" comparison since the functionality, scale, security properties, and network conditions of the two systems under test are incomparable in myriad ways.

The third test scenario, a local-area deployment of Dissent, is intended to test whether Dissent is fast enough for interactive browsing on local-area networks. The fourth scenario, a serial composition of Dissent and Tor, considers the performance of a configuration offering "best of both worlds" security, where we compose a local-area Dissent network with the public Tor network. This configuration offers users Tor's wide-area anonymity against limited-strength adversaries, combined with Dissent's local-area security against adversaries who might use traffic analysis to de-anonymize Tor circuits.

The results in Figure 10 indicate that anonymous web browsing under the local-area deployment of Dissent we tested performs comparably to Tor, suggesting that users are likely to find some Dissent configurations similarly

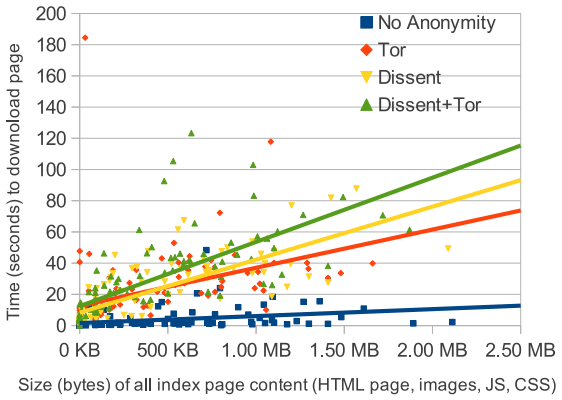

Figure 10: Download times for the Alexa "Top 100" home pages in which nodes access the Internet over an Dissent network running on an Emulab-simulated wireless LAN, over the public Tor network, and over a composition of wLAN Dissent and Tor.

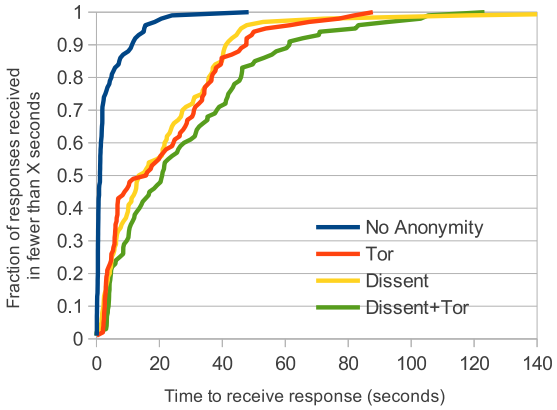

Figure 11: CDF of download times presented in Figure 10.

usable under appropriate network conditions. On average, downloading 1MB of Web content took 10 seconds with no anonymization, it took 40 seconds through Tor, 45 seconds with Dissent, and 55 seconds with Dissent and Tor together. Comparing Dissent+Tor with Tor alone, this data suggests that a user willing to tolerate a 35% slowdown could retain Tor's wide-area benefits while gaining traffic analysis resistant anonymity in the user's local area.

Figure 11 shows a CDF of page download times, showing that a client using Tor downloads the first 50% of Web pages in 15 seconds, while a client using Dissent+Tor downloads 50% of Web pages in just under 20 seconds. We expect that many users, especially those with strong security requirements, might find a few extra seconds per Web page a reasonable price for local-area security.

6 Conclusion

This paper has made the case that by delegating collective trust to a decentralized group of servers, strong anonymity techniques offering traffic analysis resistance may be adapted and scaled to offer anonymity in groups of thousands of nodes, two orders of magnitude larger than previous systems offering strong anonymity. Through its novel client/server DC-nets model, Dissent is able to accommodate anonymity set sizes of up to 5,000 members, while maintaining end-to-end latency low enough to enable wide-area interactive messaging. In local-area settings, Dissent is fast enough to handle interactive Web browsing while still offering users strong local anonymity guarantees. Although Dissent represents a step towards strong anonymous communication at large Internet scales, many challenges remain for future work, such as further scalability and robustness improvements and protection against long-term intersection attacks.

Acknowledgments

We would like to thank Michael F. Nowlan, Vitaly Shmatikov, Joan Feigenbaum, Ramakrishna Gummadi, Emin Gün Sirer, Jon Howell, Roger Dingledine, and the anonymous OSDI reviewers for their extraordinarily helpful comments, as well as the Emulab and DeterLab folks for their time and effort. This material is based upon work supported by the Defense Advanced Research Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001- 11-C-4018.

References

[1] T. G. Abbott, K. J. Lai, M. R. Lieberman, and E. C. Price. Browser-based attacks on Tor. In *PETS*, 2007.

[2] B. Adida. *Advances in cryptographic voting systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006.

[3] Alexa top 500 global sites, April 2012. `http://www.alexa.com/topsites`.

[4] Anonymizer, September 2012. `http://anonymizer.com/`.

[5] J. M. Balkin. Digital speech and democratic culture: A theory of freedom of expression for the information society. *Faculty Scholarship Series*, 2004. Paper 240.

[6] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against Tor. In *WPES*, Oct. 2007.

[7] M. Bellare and P. Rogaway. Optimal asymmetric encryption – how to encrypt with RSA. In *Eurocrypt*, May 1994.

[8] A. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2003.

[9] S. L. Blond, P. Manils, A. Chaabane, M. A. Kaafar, A. Legout, C. Castellucia, and W. Dabbous. De-anonymizing BitTorrent users on Tor, Apr. 2010. `http://arxiv.org/abs/1004.1267`.

[10] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *ACM CCS*, Oct. 2007.

[11] J. Brickell and V. Shmatikov. Efficient anonymity-preserving data collection. In *ACM KDD*, Aug. 2006.

[12] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*, Feb. 1999.

[13] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Identifying proxy nodes in a Tor anonymization circuit. In *SITIS*, Nov. 2008.

[14] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, Jan. 1988.

[15] D. Chaum and T. P. Pedersen. Wallet databases with observers. CRYPTO, 1992.

[16] B. Chun et al. PlanetLab: An overlay testbed for broad-coverage services. In *ACM CCR*, July 2003.

[17] J. Clark, P. C. van Oorschot, and C. Adams. Usability of anonymous web browsing: an examination of tor interfaces and deployability. In *SOUPS*, 2007.

[18] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[19] M. Clarkson, S. Chong, and A. Myers. Civitas: Toward a secure voting system. In *IEEE SP*, may 2008.

[20] H. Corrigan-Gibbs and B. Ford. Dissent: accountable anonymous group messaging. In *ACM CCS*, Oct. 2010.

[21] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III anonymous remailer protocol. In *IEEE SP*, May 2003.

[22] R. Deibert, J. Palfrey, R. Rohozinski, and J. Zittrain. *Access Denied: The Practice and Policy of Global Internet Filtering*. MIT Press, Jan. 2008.

[23] Deterlab network security testbed, September 2012. `http://isi.deterlab.net/`.

[24] R. Dingledine and J. Appelbaum. How governments have tried to block Tor, 2012. Tor project presentation, `https://svn.torproject.org/svn/projects/presentations/slides-28c3.pdf`.

[25] R. Dingledine and N. Mathewson. Design of a blocking-resistant anonymity system, Nov. 2006. Tor Project technical report, `https://svn.torproject.org/svn/projects/design-paper/blocking.html`.

[26] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *USENIX Security Symposium*, 2004.

[27] P. Eckersley. How unique is your web browser? In *PETS*, July 2010.

[28] M. Edman and P. Syverson. As-awareness in tor path selection. In *ACM CCS*, 2009.

[29] Emulab network emulation testbed, September 2012. `http://emulab.net/`.

[30] Federal Trade Commission. Protecting consumer privacy in an era of rapid change, Dec. 2010. Preliminary FTC Staff Report.

[31] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *ACM TOCS*, Feb. 2002.

[32] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *CRYPTO*, Aug. 2001.

[33] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *IEEE ICDCS*, june 2005.

[34] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVE: anonymous location-based queries in distributed mobile systems. In *16th WWW*, May 2007.

[35] S. Goel, M. Robson, M. Polte, and E. G. Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Technical Report 2003-1890, Cornell University, February 2003.

[36] P. Golle and A. Juels. Dining cryptographers revisited. *Eurocrypt*, May 2004.

[37] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. How much anonymity does network latency leak? In *ACM CCS*, Oct. 2007.

[38] N. Jones, M. Arye, J. Cesareo, and M. J. Freedman. Hiding amongst the clouds: A proposal for cloud-based onion routing. In *FOCI*, Aug. 2011.

[39] D. Kedogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In *5th International Workshop on Information Hiding*, Oct. 2002.

[40] S. F. Kreimer. Technologies of protest: Insurgent social movements and the First Amendment in the era of the Internet. *University of Pennsylvania Law Review*, October 2001.

[41] R. Meushaw and D. Simard. NetTop: Commercial technology in high assurance applications. *Tech Trend Notes*, 2000.

[42] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach. AP3: Cooperative, decentralized anonymous communication. In *ACM SIGOPS EW*, Sept. 2004.

[43] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy*, May 2005.

[44] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *ACM CCS*, Nov. 2001.

[45] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *WPES*, Oct. 2011.

[46] J. Preston. Facebook Officials Keep Quiet in Its Role in Revolts, Feb. 2011. `http://www.nytimes.com/2011/02/15/business/media/15facebook.html`.

[47] S. Rhea, B.-G. Chun, J. Kubiatowicz, and S. Shenker. Fixing the embarrassing slowness of OpenDHT on PlanetLab. In *WORLDS*, 2005.

[48] S. Sengupta. Rushdie runs afoul of web's real-name police. *New York Times*, Nov. 2011.

[49] E. G. Sirer et al. Eluding carnivores: File sharing with strong anonymity. In *ACM SIGOPS EW*, Sept. 2004.

[50] R. Smits et al. BridgeSPA: Improving Tor bridges with single packet authorization. In *WPES*, Oct. 2011.

[51] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle. Flash cookies and privacy, Aug. 2009.

[52] F. Stajano and R. Anderson. The cocaine auction protocol: On the power of anonymous broadcast. In *Information Hiding Workshop*, Sept. 1999.

[53] E. Stein. Queers anonymous: Lesbians, gay men, free speech, and cyberspace. *Harvard Civil Rights-Civil Liberties Law Review*, 2003.

[54] P. Syverson. Sleeping dogs lie on a bed of onions but wake when mixed. In *HotPETs*, July 2011.

[55] Tails: The amnesic incognito live system, September 2012. `https://tails.boum.org/`.

[56] A. Teich, M. S. Frankel, R. Kling, and Y. Lee. Anonymous communication policies for the Internet: Results and recommendations of the AAAS conference. *Information Society*, May 1999.

[57] E. Volokh. Freedom of speech and information privacy: The troubling implications of a right to stop people from speaking about you. *Stanford Law Review*, May 2000.

[58] M. Waidner and B. Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Eurocrypt*, Apr. 1989.

[59] J. D. Wallace. Nameless in cyberspace: Anonymity on the internet, Dec. 1999. Cato Briefing Paper No. 54.

[60] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable anonymous group communication in the anytrust model. In *EuroSec*, Apr. 2012.

[61] J. Wortham. Cellphones Now Used More for Data Than for Calls, May 2010. `http://www.nytimes.com/2010/05/14/technology/personaltech/14talk.html`.

[62] J. Wright, T. de Souza, and I. Brown. Fine-grained censorship mapping information sources, legality and ethics. In *FOCI*, Aug. 2011.