Byzantine fault-tolerant vote collection for D-DEMOS, a distributed e-voting system

Nikos Chondros*

National and Kapodistrian University of Athens
Department of Informatics and Telecommunications
n.chondros@di.uoa.gr

Abstract. E-voting systems are a powerful technology for improving democracy by reducing election cost, increasing voter participation, and even allowing voters to directly verify the entire election procedure. Unfortunately, prior internet voting systems have single points of failure, which may result in the compromise of availability, voter secrecy, or integrity of the election results.
In this thesis, we consider increasing the fault-tolerance of voting systems by introducing distributed components. This is non-trivial as, besides integrity and availability, voting requires safeguarding confidentiality as well, against a malicious adversary. We focus on the vote collection phase of the voting system, which is a crucial part of the election process.
We use the DEMOS state-of-the-art but centralized voting system as the basis for our study. This system uses vote codes to represent voters' choices, an Election Authority to setup the election and handle vote collection and result production, and a Bulletin Board for storing the election transcript for the long-term. We extract the vote collection mechanism from the centralized Election Authority component of the original DEMOS system, and replace it with a distributed system that handles vote collection in a Byzantine fault-tolerant manner. In this thesis, we present the design, security analysis, prototype implementation and experimental evaluation of this vote collection component.
We present two versions of this component: one completely asynchronous and one with minimal timing assumptions but better performance. Both versions provide immediate assurance to the voter her vote was recorded as cast, without requiring cryptographic operations on behalf of the voter, while still preserving privacy. This way, a voter may cast her vote using an untrusted computer or network, and still be assured her vote was recorded as cast. For example, she may vote via a public web terminal, or by sending an SMS from a mobile phone.

1 Dissertation Summary

1.1 Problem Description

E-voting systems are a powerful technology to improve the election process. Kiosk-based e-voting systems allow the tally to be produced faster, but require

* Dissertation Advisor: Mema Roussopoulos, Associate Professor

the voter's physical presence at the booth. Internet e-voting systems, however, allow voters to cast their votes remotely. Internet voting systems have the potential to enhance the democratic process by reducing election costs and by increasing voter participation for social groups that face considerable physical barriers and overseas voters. In addition, several internet voting systems allow voters and auditors to directly verify the integrity of the entire election process, providing *end-to-end verifiability*. This is a highly desired property that has emerged in the last decade, where voters can be assured that no entities, even the election authorities, have manipulated the election result. Despite their potential, existing internet voting systems suffer from single points of failure, which may result in the compromise of voter secrecy, service availability, or integrity of the result.

In this thesis, we consider increasing the fault-tolerance of voting systems by introducing distributed components, while still preserving privacy and end-to-end verifiability. We use the DEMOS [13] state-of-the-art but centralized voting system as the basis for our study.

In its current form, the DEMOS voting system is centralized, having an Election Authority (EA) component that handles everything from setup, to vote collection, to result production. This presents a risk to availability, as a failure of this component would prohibit voting. However, it also presents a risk to voters' privacy, as an attacker that takes control of this component can obtain each voter's ballot contents, which directly violates the voter's privacy. Finally, the original centralized DEMOS system had no need to provide feedback to the voter, besides a simple acknowledgment. In a distributed world though, the voter needs to obtain feedback to be assured the vote was actually recorded as cast in enough nodes of the system, something we tackle in this thesis.

One specific attribute of DEMOS is its use of code-voting. In this scheme, there is a setup component which generates vote codes representing the possible voter's choices, and includes them in the voters' ballots. A voter votes by submitting the vote code corresponding to her choice. Because of this technique, the voter does not need to perform cryptographic operations on the device she uses to vote. Expanding on this, we set out to introduce a distributed voting system that uses no client-side cryptography at all. This allows votes to be cast with a greater variety of client devices over public networks, such as feature phones using SMS, or (untrusted) public web terminals, while still preserving voter's privacy.

1.2 Related work

Several end-to-end verifiable e-voting systems have been introduced, e.g. the kiosk-based systems [4, 12, 3, 2, 16] and the internet voting systems [1, 14, 17, 13]. In all these works, the Bulletin Board (BB) is a single point of failure and has to be trusted.

Dini presents a distributed e-voting system, which however is not end-to-end verifiable [11]. In [9], there is a distributed BB implementation, also handling vote collection, according to the design of the vVote end-to-end verifiable e-voting system [8], which in turn is an adaptation of the Prêt à Voter e-voting system [4].

In [9], the proper operation of the *BB* during ballot casting requires a trusted device for signature verification. In contrast, our vote collection subsystem is done so that correct execution of ballot casting can be "human verifiable", i.e., by simply checking the validity of the obtained receipt. Additionally, our vote collection subsystem in D-DEMOS/Async is fully asynchronous, always deciding with exactly $n - f$ inputs, while in [9], the system uses a synchronous approach based on the FloodSet algorithm from [15] to agree on a single version of the state.

1.3 Results

In this thesis, we present the design, security analysis, prototype implementation and experimental evaluation of the vote collection components of the *D-DEMOS* [7] suite of distributed, end-to-end verifiable internet voting systems, with no single point of failure during the election process (that is, besides setup).

We design a distributed *Vote Collection* (*VC*) subsystem that is Byzantine fault-tolerant and able to collect votes from voters and assure them their vote was recorded as cast, without requiring any cryptographic operation from the client device. At election end time, *VC* nodes agree on a single set of votes.

We introduce two versions of the voting components of D-DEMOS that differ in how they achieve agreement on the set of cast votes. The D-DEMOS/Async version is completely asynchronous, while D-DEMOS/IC makes minimal synchrony assumptions but is more efficient. Once agreement has been achieved, *VC* nodes upload the set of cast votes to a second distributed component, the *Bulletin Board* (*BB*). This, in turn, is a replicated service that publishes its data immediately and makes it available to the public forever.

The resulting voting systems are end-to-end verifiable, by the voters themselves and third-party auditors, while preserving voter privacy. To delegate auditing, a voter provides an auditor specific information from her ballot. The auditor, in turn, reads from the distributed *BB* and verifies the complete election process, including the correctness of the election setup by election authorities. Additionally, as the number of auditors increases, the probability of election fraud going undetected diminishes exponentially.

The thesis is structured as follows. Section 1 provides an introduction to the problem, and gives a short description of DEMOS, the system we use as a model and we extend to become fault-tolerant. Section 2 gives background information on voting systems (using information from [5], and tools from distributed systems and cryptography that we employ in our system designs.

Section 3 provides a thorough system description of the two systems we build. It first gives the system model, and the gradually introduces the Vote Collection subsystems we introduce, a mostly-asynchronous one with a single timing assumption (for D-DEMOS/IC) and a completely asynchronous design (for D-DEMOS/Async). It also describes how the systems are initialized by the EA component, and how the voter uses our system to vote. This section also includes the proofs of liveness and safety for both vote collection approaches. In both approaches to vote collection, we design a *voting protocol* that is active

during voting hours and collects votes from the voters, and a *vote set consensus* protocol that ensures agreement between vote collection nodes after voting is finished, and allows the system to progress towards producing the election result.

Section 4 provides answers to questions regarding our system design. First of all, it answers why standard approaches, like a Byzantine Fault Tolerant Replicated State Machine (such as [6]), is not suitable for the problem at hand. It then lists a series of possible attack vectors from different system components, and describes how our system thwarts them.

Section 5 outlines our prototype implementation. It describes our message-passing substrate and its interaction with our Web front-ends, and our implementation of the EA and both versions of the Vote Collection subsystems. It also describes our implementation of Bracha's asynchronous binary consensus, that is used in the asynchronous version of the system.

Section 6 describes the evaluation and presents our experimental results. We perform experiments with a relational database as a data store, and also with an in-memory data storage approach. We perform experiments on a LAN, and we also simulate a WAN. The outcome of the evaluation is that the D-DEMOS/IC vote collection approach is slightly faster during voting (around 15%), and quite faster during vote set consensus (4 times faster). for the disk-based experiments.

2 Vote collection for D-DEMOS

We will now briefly describe the design of our Vote Collection (*VC*) subsystem.

We design the *VC* subsystem as a distributed system of N_v cooperating nodes, tolerating up to f_v Byzantine faults, where $f_v < N_v/3$. Note that, we also tolerate the collusion of an arbitrary number of malicious voters with the malicious *VC* nodes. *VC* nodes have private communication channels to each other, and a public (unsecured) channel for the voters.

We modify the data generation process of DEMOS's *EA*, by adding the following two steps while generating voter's ballots:

1. The (random) vote-code corresponding to each election option is provided in committed form to each *VC* node.
2. A receipt is generated for each vote code, which is itself a random number. The receipt is secret shared across *VC* nodes with a Verifiable Secret Sharing (VSS) scheme. Each *VC* node receives one of these shares.

At step 1, the commitment scheme used hashes the plain text message along with a salt. The salt is provided along with the committed form to each *VC* node, while the opening of the commitment is the vote-code itself.

Before going into detail in the design of the Vote Collection subsystem, we give an overview of its use. *VC* nodes are initialized from the *EA* (as above). The voter receives her ballot also from the *EA*, along with the addresses of the *VC* nodes. During the election hours, *VC* nodes run the *voting protocol*.

For this protocol to start, the voter selects one part of her ballot at random, and posts her selected vote code to one of the *VC* nodes. The *VC* node that

receives her vote validates it, interacts with the other VC nodes to reconstruct the receipt from the shares spread across the VC nodes, and posts it back to the voter. When she receives a receipt, she compares it with the one on her ballot corresponding to the selected vote code. If it matches, she is assured her vote was correctly recorded and will be included in the election tally. The other part of her ballot, the one not used for voting, will be used for auditing purposes. This design is essential for verifiability, in the sense that the EA cannot predict which part a voter may use, and the unused part will betray a malicious EA with $\frac{1}{2}$ probability per audited ballot.

At election end time, VC nodes run our Vote Set Consensus protocol, which guarantees all VC nodes agree on a single set of voted vote codes. After agreement, each VC node uploads this set to every BB node, which in turn publishes this set once it receives the same copy from enough $(f_v + 1)$ VC nodes.

2.1 Synchronous version, for D-DEMOS/IC

The *voting* protocol starts when a voter submits a VOTE⟨serial-no, vote-code⟩ message to a VC node. We call this node the *responder*, as it is responsible for delivering the receipt to the voter. The VC node confirms the current system time is within the defined election hours, and locates the ballot with the specified serial-no. It also verifies this ballot has not been used for this election, either with the same or a different vote code. Then, it compares the vote-code against every hashed vote code in each ballot line, until it locates the correct entry. Subsequently, it obtains from its local database the receipt-share corresponding to the specific vote-code. Next, it marks the ballot as pending for the specific vote-code. Finally, it multicasts a VOTE_P⟨serial-no, vote-code, receipt-share⟩ message to all VC nodes, disclosing its share of the receipt. In case the located ballot is marked as voted for the specific vote-code, the VC node sends the stored receipt to the voter without any further interaction with other VC nodes.

Each VC node that receives a VOTE_P message, first validates the received receipt-share according to the verifiable secret sharing scheme used. Then, it performs the same validations as the responder, and multicasts another VOTE_P message (only once), disclosing its share of the receipt. When a node collects $h_v = N_v - f_v$ valid shares, it uses the verifiable secret sharing reconstruction algorithm to reconstruct the receipt (the secret) and marks the ballot as voted for the specific vote-code. Additionally, the *responder* node sends this receipt back to the voter.

A message flow diagram of our *voting* protocol is depicted in Figure 1. As is evident from the diagram, the time from the multicast of the first VOTE_P message until collecting all receipt shares, is only slightly longer than a single round-trip between two VC nodes.

At election end time, each VC node stops processing VOTE and VOTE_P messages, and initiates the *vote-set consensus* protocol. It creates a set VS_i of ⟨serial-no, vote-code⟩ tuples, including all *voted* and *pending* ballots. Then, it participates in the Interactive Consistency (IC) protocol of [10], with this set. At the end of IC, each node contains a vector $\langle VS_1, \ldots, VS_n \rangle$ with the Vote

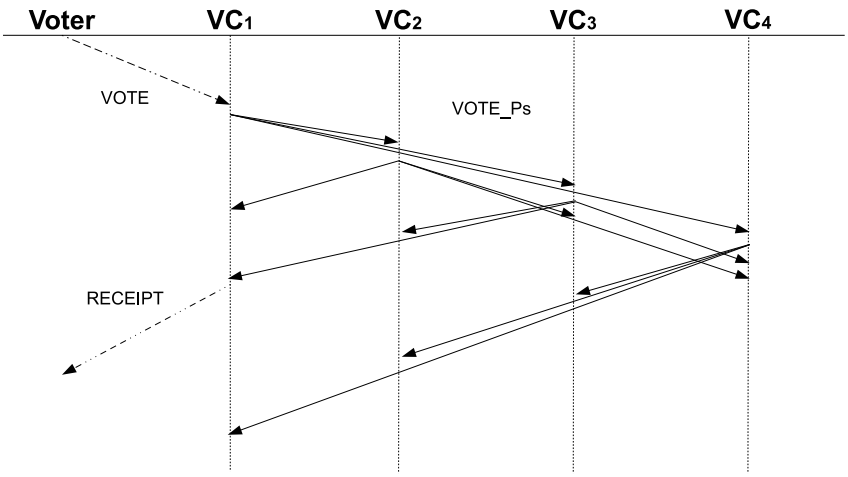

Fig. 1. Diagram of message exchanges for a single vote during the D-DEMOS/IC vote collection phase.

Set of each node, and follows the algorithm of Figure 2. Step 1 makes sure

Cross-tabulate $\langle VS_1, \ldots, VS_n \rangle$ per ballot, creating a list of vote codes for each ballot. Perform the following actions for each ballot:

1. If the list contains two or more distinct vote codes, mark the ballot as NotVoted and exit.
2. If a vote code vc_a appears at least $N_v - 2f_v$ times in the list, mark the ballot as Voted for vc_a and exit.
3. Otherwise, mark the ballot as NotVoted and exit.

Fig. 2. High level description of algorithm after IC.

any ballot with multiple submitted vote codes is discarded. Since vote codes are private, and cannot be guessed by malicious vote collectors, the only way for multiple vote codes to appear is if malicious voters are involved, against whom our system is not obliged to respect our *contract*.

With a single vote code remaining, step 2 considers the threshold above which to consider a ballot as voted for a specific vote code. We select the $N_v - 2f_v$ threshold for which we are certain that even the following extreme scenario is handled. If the *responder* is malicious, submits a receipt to an honest voter, but denies it during *vote-set consensus*, the remaining $N_v - 2f_v$ honest *VC* nodes that revealed their receipt shares for the generation of the receipt, are enough for the system to accept the vote code (receipt generation requires $N_v - f_v$ nodes, of which f_v may be malicious, thus $N_v - 2f_v$ are necessarily honest).

Finally, step 3 makes sure vote codes that occur less than $N_v - 2f_v$ times are discarded. Under this threshold, there is no way a receipt was ever generated.

At the end of this algorithm, each node submits the resulting set of *voted* $\langle$serial-no, vote-code$\rangle$ tuples to each *BB* node, which concludes its operation for the specific election.

2.2 Asynchronous version, for D-DEMOS/Async

We make the following enhancements to the Vote Collection subsystem, to achieve the completely asynchronous version *D-DEMOS/Async*. During voting we introduce another step, which guarantees only a single vote code can be accepted (towards producing a receipt) for a given ballot. We also employ an asynchronous binary consensus primitive to achieve Vote Set Consensus.

More specifically, during voting, the *responder VC* node validates the submitted vote code, but before disclosing its receipt share, it multicasts an ENDORSE$\langle$serial-no, vote-code$\rangle$ message to all *VC* nodes. Each *VC* node, after making sure it has not endorsed another vote code for this ballot, responds with an ENDORSEMENT$\langle$serial-no, vote-code, $\mathsf{sig}_{\mathsf{VC_i}}\rangle$ message, where $\mathsf{sig}_{\mathsf{VC_i}}$ is a digital signature of the specific serial-no and vote-code, with VC_i's private key. The responder collects $N_v - f_v$ valid signatures and forms a uniqueness certificate UCERT for this ballot. It then discloses its receipt share via the VOTE_P message, but also includes the formed UCERT in the message.

Each *VC* node that receives a VOTE_P message, first verifies the validity of UCERT and discards the message on error. On success, it proceeds as per the *D-DEMOS/IC* protocol (validating the receipt share it receives and then disclosing its own receipt share).

The voting process is outlined in the diagram of Figure 3, where we now see two round-trips are needed before the receipt is reconstructed and posted to the voter.

The formation of a valid UCERT gives our algorithms the following guarantees:

a) No matter how many responders and vote codes are active at the same time for the same ballot, if a UCERT is formed for vote code vc_a, no other uniqueness certificate for any vote code different than vc_a can be formed.
b) By verifying the UCERT before disclosing a *VC* node's receipt share, we guarantee the voter's receipt cannot be reconstructed unless a valid UCERT is present.

At election end time, each *VC* node stops processing ENDORSE, ENDORSEMENT, VOTE and VOTE_P messages, and follows the *vote-set consensus* algorithm in Figure 4, for each registered ballot.

Steps 1-2 ensure used vote codes are dispersed across nodes. Recall our receipt generation requires $N_v - f_v$ shares to be revealed by distinct *VC* nodes, of which at least $N_v - 2f_v$ are honest. Note that any two $N_v - f_v$ subsets of N_v contain at least $f_v + 1$ honest nodes (because $f_v > N_v/3$), and at least one of the $f_v + 1$

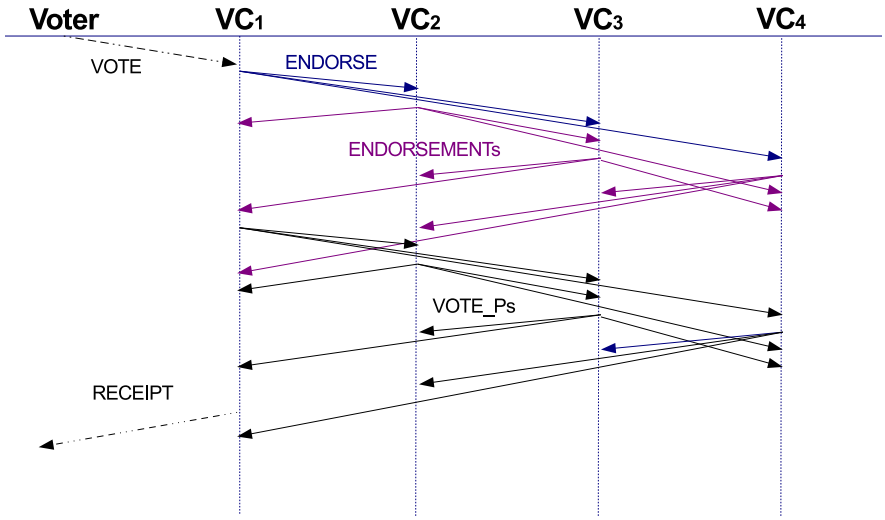

Fig. 3. Diagram of message exchanges for a single vote during the D-DEMOS/Async vote collection phase.

1. Send `ANNOUNCE`⟨serial-no, vote-code, UCERT⟩ to all nodes. The vote-code will be *null* if the node knows of no vote code for this ballot.
2. Wait for $N_v - f_v$ such messages. If any of these messages contains a valid vote code vc_a, accompanied by a valid UCERT, change the local state immediately, by setting vc_a as the vote code used for this ballot.
3. Participate in a Binary Consensus protocol, with the subject "Is there a valid vote code for this ballot?". Enter with an opinion of 1, if a valid vote code is locally known, or a 0 otherwise.
4. If the result of Binary Consensus is 0, consider the ballot not voted.
5. Else, if the result of Binary Consensus is 1, consider the ballot voted. There are two sub-cases here:
   a) If vote code vc_a, accompanied by a valid UCERT is locally known, consider the ballot voted for vc_a.
   b) If, however, vc_a is not known, send a `RECOVER-REQUEST`⟨serial-no⟩ message to all VC nodes, wait for the first valid `RECOVER-RESPONSE`⟨serial-no, vc_a, UCERT⟩ response, and update the local state accordingly.

Fig. 4. High level description of algorithm for asynchronous vote set consensus. This algorithm runs for each registered ballot.

honest nodes has participated in receipt generation. Because of this, if a receipt was generated, at least one honest node's `ANNOUNCE` will be processed by every honest node, and all honest VC nodes will obtain the corresponding vote code in these two steps. Consequently, all honest nodes enter step 3 with an opinion of 1 and binary consensus is guaranteed to deliver 1 as the resulting value, thus

safeguarding our contract against the voters. In any case, step 3 guarantees all *VC* nodes arrive at the same conclusion, on whether this ballot is voted or not.

In the algorithm outlined above, the result from binary consensus is translated from 0/1 to a status of "not-voted" or a unique valid vote code, in steps 4-5. Step 5b requires additional explanation. Assume, for example, that a voter submitted a valid vote code vc_a, but a receipt was not generated before election end time. In this case, an honest vote collector node VC_i may not be aware of vc_a at step 3, as steps 1-2 do not make any guarantees in this case. Thus, VC_i may rightfully enter consensus with a value of 0. However, when honest nodes' opinions are mixed, the consensus algorithm may produce either 0 or 1. In case the result is 1, VC_i will not possess the correct vote code vc_a, and thus will not be able to properly translate the result. Thus we introduce a recovery protocol with which VC_i will issue a `RECOVER-REQUEST` multicast. We claim that another honest node, VC_h, exists that *possesses* vc_a and *replies* with vc_a and the correct UCERT. The reason for the existence of an honest VC_h is straightforward and stems from the properties of the binary consensus problem definition. If all honest nodes enter binary consensus with the same opinion a, the result of any consensus algorithm is guaranteed to be a. Since we have an honest node VC_i, that entered consensus with a value of 0, but a result of 1 was produced, there has to exist another honest node VC_h that entered consensus with an opinion of 1. Since VC_h is honest, it must *possess* vc_a, along with the corresponding UCERT (as no other vote code vc_b can be active at the same time for this ballot). Again, because VC_h is honest, it will follow the protocol and *reply* with a well formed RECOVER-REPLY. Additionally, the existence of UCERT guarantees that any malicious replies can be safely identified and discarded by VC_i.

As per *D-DEMOS/IC*, at the end of this algorithm, each node submits the resulting set of *voted* $\langle \text{serial-no}, \text{vote-code} \rangle$ tuples to each *BB* node, which concludes its operation for the specific election.

2.3 Evaluation

We will now outline some notable results from our evaluation. In Figure 5, we plot the average response time of both our vote collection protocols, versus the number of vote collectors, under different concurrency levels, ranging from 500 to 2000 concurrent clients. The experiment is run with our in-memory data structure, highlighting the performance of our network protocols. Results for both systems illustrate an almost linear increase in the client-perceived latency, for all concurrency scenarios, up to 13 *VC* nodes. D-DEMOS/IC has a slower response time with its single round intra-*VC* node communication, while D-DEMOS/Async is slightly slower due to the extra Uniqueness Certificate round.

Figure 6 shows the throughput of both our vote collection protocols, versus the number of vote collectors, under different concurrency levels, for the same experiment. We observe that, in terms of overall system throughput, the penalty of tolerating extra failures (increasing the number of vote collectors) manifests early on. We notice an almost 50% decline in system throughput from 4 to 7 *VC* nodes

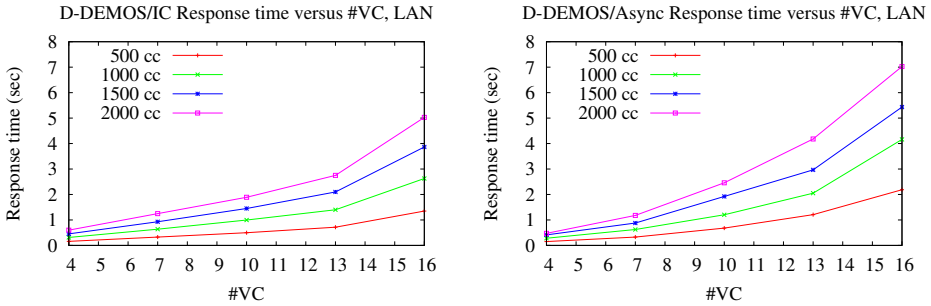

Fig. 5. Vote Collection response time of D-DEMOS/IC (5a) and D-DEMOS/Async (5b), versus the number of *VC* nodes, under a LAN setting. Election parameters are $n = 200{,}000$ and $m = 4$.

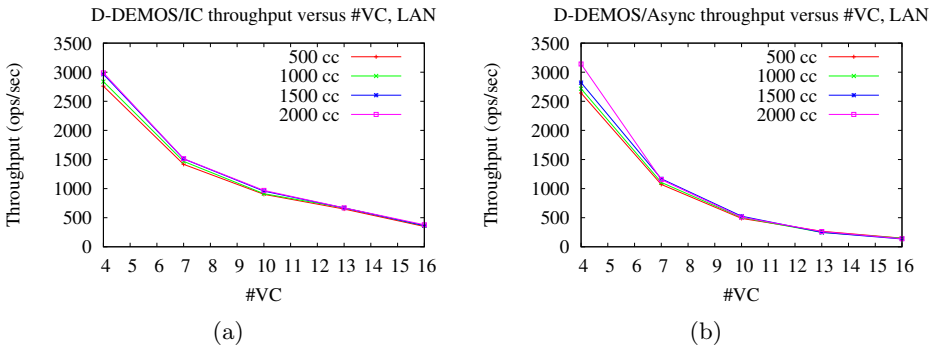

Fig. 6. Vote Collection throughput of D-DEMOS/IC (6a) and D-DEMOS/Async (6b), versus the number of *VC* nodes, under a LAN setting. Election parameters are $n = 200{,}000$ and $m = 4$.

for D-DEMOS/IC, and a bigger one for D-DEMOS/Async. However, further increases in the number of vote collectors lead to a much smoother, linear decrease. Overall, D-DEMOS/IC achieves better throughput than D-DEMOS/Async, due to exchanging fewer messages and lacking signature operations.

3 Conclusion

In this thesis, we presented two different vote collection subsystems for the D-DEMOS suite of distributed vote collection systems. Both resultant voting systems allow voters to verify their vote was tallied-as-intended without the assistance of special software or trusted devices, while maintaining the end-to-end verifiability required for external auditors to verify the correctness of the complete election process. We proved the safety and liveness of both vote collection subsystems, produced prototypes implementing them, measured their performance, and demonstrated their ability to handle large-scale elections.

We believe our vote collection subsystems are applicable to any voting system that uses the code-voting technique. Thus, we believe our work is a required step towards producing higher quality voting systems that can handle large-scale elections efficiently and reliably.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: Proceedings of the 17th USENIX Security Symposium, San Jose, CA, USA. pp. 335–348. USENIX Association (July 2008)
2. Benaloh, J., Byrne, M.D., Eakin, B., Kortum, P.T., McBurnett, N., Pereira, O., Stark, P.B., Wallach, D.S., Fisher, G., Montoya, J., Parker, M., Winn, M.: STAR-vote: A secure, transparent, auditable, and reliable voting system. In: Proceedings of the Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13, Washington, D.C., USA. USENIX Association (August 2013)
3. Chaum, D., Essex, A., Carback, R., Clark, J., Popoveniuc, S., Sherman, A., Vora, P.: Scantegrity: End-to-end voter-verifiable optical-scan voting. Security & Privacy, IEEE 6(3), 40–46 (2008)
4. Chaum, D., Ryan, P.Y.A., Schneider, S.A.: A practical voter-verifiable election scheme. In: Proceedings of the 10th European Symposium on Research in Computer Security - ESORICS 2005, Milan, Italy. pp. 118–139. Springer (September 2005)
5. Chondros, N., Delis, A., Gavatha, D., Kiayias, A., Koutalakis, C., Nicolacopoulos, I., Paschos, L., Roussopoulos, M., Sotirelis, G., Stathopoulos, P., Vasilopoulos, P., Zacharias, T., Zhang, B., Zygoulis, F.: Electronic voting systems - from theory to implementation. In: E-Democracy, Security, Privacy and Trust in a Digital World. pp. 113–122 (Dec 2013)
6. Chondros, N., Kokordelis, K., Roussopoulos, M.: On the practicality of practical byzantine fault tolerance. In: Proceedings of the ACM/IFIP/USENIX 13th International Middleware Conference (Middleware 2012), Montreal, QC, Canada. pp. 436–455. Springer (December 2012)

7. Chondros, N., Zhang, B., Zacharias, T., Diamantopoulos, P., Maneas, S., Patsonakis, C., Delis, A., Kiayias, A., Roussopoulos, M.: D-demos: A distributed, end-to-end verifiable, internet voting system. In: Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on (Jun 2016)
8. Culnane, C., Ryan, P.Y.A., Schneider, S., Teague, V.: vvote: A verifiable voting system. ACM Transactions on Information and System Security 18(1), 3:1–3:30 (Jun 2015), http://doi.acm.org/10.1145/2746338
9. Culnane, C., Schneider, S.: A peered bulletin board for robust use in verifiable voting systems. In: Proceedings of the IEEE 27th Computer Security Foundations Symposium (CSF 2014), Vienna, Austria. pp. 169–183. IEEE (July 2014)
10. Diamantopoulos, P., Maneas, S., Patsonakis, C., Chondros, N., Roussopoulos, M.: Interactive consistency in practical, mostly-asynchronous systems. In: Proceedings of the IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS 2015). pp. 752–759. IEEE (Dec 2015)
11. Dini, G.: A secure and available electronic voting service for a large-scale distributed system. Future Generation Computer Systems 19(1), 69–85 (2003)
12. Fisher, K., Carback, R., Sherman, A.: Punchscan: introduction and system definition of a high-integrity election system. In: IAVoSS Workshop On Trustworthy Elections (WOTE 2006), Cambridge, United Kingdom (June 2006)
13. Kiayias, A., Zacharias, T., Zhang, B.: End-to-end verifiable elections in the standard model. In: Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Advances in Cryptology - EUROCRYPT 2015, Sofia, Bulgaria. pp. 468–498. Springer (April 2015)
14. Kutylowski, M., Zagórski, F.: Scratch, click & vote: E2E voting over the internet. In: Towards Trustworthy Elections, New Directions in Electronic Voting, Lecture Notes in Computer Science, vol. 6000, pp. 343–356. Springer (2010)
15. Lynch, N.: Distributed Algorithms. Morgan Kaufmann (1996)
16. Moran, T., Naor, M.: Split-ballot voting: Everlasting privacy with distributed trust. ACM Transactions on Information and System Security 13(2), 16:1–16:43 (Mar 2010), http://doi.acm.org/10.1145/1698750.1698756
17. Zagórski, F., Carback, R.T., Chaum, D., Clark, J., Essex, A., Vora, P.L.: Remotegrity: Design and use of an end-to-end verifiable remote voting system. In: Proceedings of the 11th International Conference on Applied Cryptography and Network Security, ACNS 2013, Banff, AB, Canada. pp. 441–457. Springer (June 2013)