Secure Algorithms and Data Structures for Massive Networks

Jared Saia

Joint work with: Amos Fiat(U. Tel Aviv), Valerie King(U. Vic), Erik Vee (IBM Labs), Vishal Sanwalani(U. Waterloo), and Maxwell Young(UNM)
What is Security?

**Security**: Designing algorithms and data structures which are **provably** robust to attack

- **Attack**: An adversary controls a constant fraction of nodes in the network
- **Robust**: Critical invariants provably maintained despite efforts of adversary to disrupt them
Our Adversary

- Controls constant fraction of the nodes in the network
- Mostly Omniscient
- Computationally unbounded
Scalable Security

- In massive networks, number of nodes, n, can be millions
- Thus want *scalable* algorithms:
  - Bandwidth: each node can send and process only polylog n bits
  - Latency: polylog n
  - Memory: polylog n
Outline

- Motivation
- Our Results, Scalable and Secure:
  - Data Structures
    - Distributed Hash Table
  - Algorithms
    - Leader Election, Byzantine Agreement, Global Coin Toss
- Future Work
Motivation

- Scalability: Peer-to-peer, ad hoc, wireless networks can have hundreds of thousands of nodes
- Security: These networks are vulnerable
  - No admission control
  - Economic, social and political incentives to attack
  - Adversary can take over many, many nodes (e.g. zombienets) and use them maliciously
Motivation

Why computationally-unbounded adversary?

- Dangerous to assume certain problems are intractable (e.g. quantum computation can solve factoring)
- Many real-world adversaries have access to significant computational and/or human resources (e.g. governments, companies, zombienets)
- Theoretically interesting
A distributed hash table (DHT) is a structured peer-to-peer network that provides:

- Content storage and lookup

Many DHTs: Chord, CAN, Tapestry, Viceroy, Khorde, Kelips, Kademlia, etc.

We focus on Chord
Chord

- Each peer in Chord has an ID which locates it on the unit circle.
- Each peer maintains links to peers at geometrically increasing distances from itself.
- Thus, each peer can reach any other peer in $O(\log n)$ hops while maintaining $O(\log n)$ links.
Chord

- **Successor protocol enables storage and lookup of data items**

- For a point $k$, \( \text{successor}(k) \) returns peer, $p$, which minimizes clockwise distance between $k$ and $p$.

- If $k$ is the key for a data item; \( \text{successor}(k) \) is the peer that stores that data item.
Introducing: Coyotus Adversarius

An adversary can: spam, hog bandwidth, delete nodes, etc.
Chord is Vulnerable

- Chord is robust to random node deletion. But it is not robust to adversarial attack.

Adversarial peers might:
- not forward requests
- corrupt data
- etc.
Our Goals

Design variant of Chord which is:

- **Robust**: ensure correctness of successor protocol even under attack

- **Scalable**: bandwidth, latency and memory are polylogarithmic in the network size
Theorem: S-Chord is robust, whp, for any time period during which:

- there are always \( z \) peers in the network for some integer \( z \)
- there are never more than \((1/4-\varepsilon)z\) adversarial peers in the network for positive \( \varepsilon \)
- number of peer insertions and deletions is no more than \( z^k \) for some tunable parameter \( k \)
Our Result

- **Robust:**
  - Correctness of successor protocol guaranteed

- **Scalable:**
  - Resources required by S-Chord are only a polylogarithmic factor greater than Chord in bandwidth, latency, and linking costs

- **Assumption:**
  - Every direct link is a private communication channel
Main Idea: Trustworthy Small Sets

- For point $x$ on the unit circle, define the *swarm*, $S(x)$, to be set of peers whose ID's are located within clockwise distance of $\Theta((\ln n)/n)$ from $x$.
Swarm Links

Whenever a peer $p$ links to a single peer in Chord, $p$ links to a set of $O(\log n)$ peers (a swarm) in S-Chord.
Swarm Goodness Invariant

- Call a swarm *good* if it contains at least a $\frac{3}{4}$ fraction of good peers and $\Theta(\log n)$ peers total.
- Critical invariant maintained by our DHT is that all swarms are good.
- We can use this invariant to implement the *successor* protocol robustly, using majority filtering.
Successor

- If All Swarms are good, can robustly implement Successor with majority filtering
- Takes $O(\log^3 n)$ messages naively

Our improvements:
- Can do in $O(\log^2 n)$ messages in expectation
- Can also do with $O(1)$ bit blowup in expectation using Rabin fingerprinting and error-correcting codes
Join Protocol

- Join protocol maintains Swarm Goodness Invariant
- When a peer joins, it must establish its own links and links of other peers must be updated too
- We assume that a joining peer knows some good peer in the network
Join Protocol

- Adversary selects IP addresses so we can’t use these to determine proper location for a peer in our DHT.
- Thus, when a new peer \( p \) joins the network, it is assigned an ID by a pre-existing swarm \( S \) in the network.
- \( S \) needs a way to come to consensus on the ID of \( p \).
Selecting a random ID

- Use techniques from secure multiparty computation to allow a good swarm $S$ to agree on a random number between 0 and 1

- Can do this even if a computationally unbounded adversary controls a $1/4$ fraction of the peers in the swarm

- Requires private communication channels between all peers in the swarm
A Problem

- Random ID selection will insert bad peers at random locations
- However, adversary can target a swarm and keep adding peers to the network, discarding those that land outside the targeted swarm, until there is a majority of bad peers in that swarm
- Adversary only needs to add $O(z)$ peers before it will be able to take over some swarm
Solution

- [S ’05] shows that if each joining peer is rotated with two other peers selected u.a.r. that the bad peers will be sufficiently scattered so that they can not take over a swarm (for $Z^k$ insertions)

- [KS ’04] give an algorithm for selecting a peer u.a.r. from the set of all peers in a DHT. Algorithm can be run by a good swarm to come to consensus on two peers selected u.a.r.

- Combining these two results allows us to maintain the Swarm Goodness Invariant w.h.p for $Z^k$ peer joins.
Join Protocol

- The JOIN algorithm assumes that peer $p$ knows some correct peer $q$

- $p$ first contacts peer $q$ with a request to join the network.

- $q$ alerts $S(q)$ to this request and the peers in $S(q)$ choose a random ID for $p$ using secure computation.

- Two peers, $p_1$ and $p_2$, are then selected uniformly at random and then $p$, $p_1$, and $p_2$ are rotated.
All swarms are good - Pf Intuition

- Good peers are “well spread” on the unit circle since their lifetimes are independent of locations.
- Whenever a new peer is added, there is a small random perturbation of the peer locations on the unit circle.
- This ensures that the bad peers are also well spread on the circle.
- Thus every swarm has a majority of good peers.
Handling different estimates

- So far we have assumed that all peers know $\ln n$ and $(\ln n)/n$ exactly – this is clearly unrealistic.

- However, using standard techniques, we can ensure that each peer has high and low estimates of these quantities.

- Using these estimates, the protocols remain essentially the same and all previous results hold.
DHT Conclusion

- S-Chord provably preserves functionality of Chord even in the face of massive adversarial attack.

- For n peers in the network, the resource costs are:
  - $O(\log n)$ latency and expected $\Theta(\log^2 n)$ messages per lookup
  - $\Theta(\log n)$ latency and $\Theta(\log^3 n)$ messages per peer join operation
  - $O(\log^2 n)$ links stored at each peer
Outline

- Motivation
- Our Results, Scalable and Secure:
  - Data Structures
    - Distributed Hash Table
  - Algorithms
    - Leader Election, Byzantine Agreement, Global Coin Toss
- Future Work
Leader Election

- In the leader election problem, there are n processors, 1/3 of which are bad.
- Bad processors are controlled by an adversary which selects them before game starts.
- Goal: Design algorithm which ensures a good processor is elected with constant probability.
Leader Election

- Communication occurs in rounds, bad processors get to see messages of all good players before they send their messages.
- Every processor has a unique ID - the ID of the sender of a message is explicitly known by the receiver.
- Each processor has access to private random bits which are not known to the adversary or the other processors.
Our Goal

- Previous results: can solve this problem in small number of rounds but require that each processor send and process a number of bits which is linear in $n$.

- Our goal: an algorithm which is scalable: each good processor sends and processes a number of bits which is at most polylogarithmic in $n$ (exponential decrease).
**Our Result**

- Assume there are $n$ processors and strictly less than $1/3$ are bad. Our algorithm elects, with constant probability, a leader from the set of good processors such that:
  - Exactly one good processor considers itself the leader
  - A $1-o(1)$ fraction of the good processors know this leader
  - Every good processor sends and processes only a polylogarithmic number of bits
  - The number of rounds required is polylogarithmic in $n$
Sampling

- Result: almost all \((1-o(1)\) fraction) of the good processor know the leader
- Using sampling, we can bootstrap this to ensure that w.h.p, all good processors know the leader
- However can only do this if
  - Have private communication channels
  - Restrict number of messages bad nodes can send
Techniques Used

- Our algorithm makes use of a “tournament” graph which has expander-like properties.
- Each bottom node of this graph corresponds to a small set of randomly chosen processors.
- Processors advance up the graph as they win local elections.
Q: How to ensure that the winner of some lower election knows its competitors at the next higher election?

A: Idea: Use watcher sets: sets of nodes that watch an election but don’t participate.

Hard part: setting up these watcher sets so that most of them can’t be taken over by the adversary.
Extensions

- We can easily extend our result to elect with a set of $O(\log n)$ processors such that with high probability, a majority of these peers are good.

- This allows us to securely compute several other problems w.h.p. e.g., majority vote, Byzantine agreement, etc.
We’ve described provable secure and scalable

- Data Structures: Distributed Hash Table (DHT)
- Algorithms: Leader Election, Byzantine Agreement, Global Coin Toss

Our algorithms are robust against a computationally unbounded, omniscient adversary that controls a constant fraction of the network
Future Work

- Robustification: Can we take other algorithms and make them robust without blowing up number of messages by too much?
  - E.g. Worm detection, Collaborative Filtering, Auctions, Voting Protocols, Spectral Decomposition

- Practical Applications: Can we simplify the algorithms enough so they can be successfully deployed on real networks?
That’s all folks!
Related Work

- Several results deal with Byzantine attacks on p2p networks.
- Common model: single attack where each peer independently has probability $p<1/2$ of becoming Byzantine. [FS ’02, NW ‘03, HK ‘04]
  - Problem: more likely scenario is many Byzantine peers joining the network over time
- Awerbuch and Scheideler [AS ‘04] design a secure distributed naming service which is robust to multiple Byzantine attacks
  - Problem: requires every peer to rejoin the network after $O(\log n)$ time steps
  - Their system is not a DHT (it is a distributed naming service)
Join Protocol

- All peers in $S(p)$ find all the peers in $p$'s Forward and Backward intervals

- In addition, the peers in $S(p)$ introduce $p$ to all peers, $p'$, in the network such that $p$ is now in a Center, Forward or Backward interval for $p'$

- In a similar fashion $p_1, p_2$ are rotated into their new positions and their new Center, Forward, and Backward intervals are established

- JOIN protocol requires $O(\log n)$ latency and $O(\log^3 n)$ messages
Join Protocol
P2P Future Work

- We conjecture that these techniques can be extended to a number of other ring-based DHTs that have a finger-function $f$ which satisfies $|f(x) - f(x+\delta)| \leq \delta$ for all positive $\delta$ and any point $x$ on the unit circle.
- Can these protocols or heuristics based on them be used in a practical p2p system? How can the protocols be simplified?
- Can we improve upon the message complexity for the robust successor protocol? Is it possible to either get less than $O(\log^2 n)$ expected messages or prove that this is not possible?