IN2194
Peer-to-Peer Systems and Security
Summer Semester 2023

Midterm Report

Team Gossiphers (Gossip-1):
Yannis Matezki and Dominik Ritzenhoff

Date Submitted: Jul. 04, 2023

Changes to Initial Assumption

Use of external libraries:

- go.uber.org/zap - Structured Logging library for more convenient logging
- gopkg.in/ini.v1 - Ini Parser for the given configuration file

All other original assumptions from the Initial Report still stand

Architecture of the Module

Logical Structure

Go does not offer classes, but structs with method implementations offer a similar functionality.
Our module is separated into a few logical units:

- API
    - TCP Server that answers API requests defined in the given specification
    - API Packet serialization/deserialization
    - Basic logic for API clients to listen to events of a certain type
    - Structs used like classes:
        - *Server*
            - Stateful TCP API server, stores connections registered for events
- Config
    - Configuration values are read from a .ini file, which serve as adjustable parameters within the program.
- Gossip
    - TCP Server and Client that answer and initiate P2P push and pull requests defined within the Brahms algorithm, respectively.
    - P2P Packet serialization/deserialization.
    - Structs used as classes:
        - *View* - Represents a view within Brahms algorithm.
        - *Sampler* - Represents a sampler within Brahms algorithm.
        - *SamplerGroup* - Represents a group of samples belonging to one node.
        - *Node* - Represents a node (i.e. peer) within the network.
        - *Gossip* - Represents the collection of data structures needed to make the Brahms algorithm function.
        - *Server* - Stateful TCP P2P API server. Stores connections registered for events.
- Challenge
    - Module to generate computational challenges in the required format and check the validity of the produced solutions
    - Utilized a ring of rotating keys to invalidate old challenges after a given timeframe without storing each generated challenge
    - Structs used like classes:
        - *Challenger*
            - Stores internal key ring and automatically rotates keys in a given time frame
            - Generates challenges and validates solutions using the internal keys

Process Architecture

Goroutines are utilized for all internal processes acting as lightweight threads managed by the Go runtime. Each TCP server is running in its own Goroutine and spawns a new Goroutine for each incoming connection to allow for multiple parallel connections. Goroutines are also utilized for periodically executed tasks like the key rotation of the challenge generator.

Networking

For the P2P protocol, our design was strongly influenced by the API protocol with respect to the way we parse the packets.
Packets that are not formatted correctly are simply dropped, a flooding of well formatted packets is currently not prevented, but the local view should not be impacted due to other security measures.
A broken connection does not have a lasting impact on communication, as all communication between nodes is stateless, though any dropped packets might lead to a node having a degraded connectivity on the network as unresponded pings will lead to the removal from other peers' views.
Each row in one of the following tables is 32 bits wide.

Gossip Ping:

- Sent to probe another node, other node should reply with a GOSSIP PONG if available

size	GOSSIP PING = 0x00 0x30

Gossip Pong:

- A response to the ping, indicating that this node is still alive.

size	GOSSIP PONG = 0x00 0x31

Gossip Pull Request:

- Sent to request a list of nodes from another node, the other node should respond with a GOSSIP PULL RESPONSE

size	GOSSIP PULL REQUEST = 0x00 0x40

Gossip Pull Response:

- Response to the pull request with the node’s view included in the body.

size	GOSSIP PULL RESPONSE = 0x00 0x41
nodes (comma concatenated string)	

Gossip Push Request:

- Request a challenge to push the local view to another node, other node should respond with a GOSSIP PUSH CHALLENGE to this

size	GOSSIP PUSH REQUEST = 0x00 0x51

Gossip Push Challenge:

- Response to the push request. Includes a challenge token for the destination node to ‘solve’ in order to have its gossip push request acknowledged.

size	GOSSIP PUSH CHALLENGE = 0x00 0x52
challenge	

Gossip Push:

- After solving the challenge received in a GOSSIP PUSH CHALLENGE packet, a node can then continue to send a list of nodes from the local view to another node with the nonce that acts as the solution to the challenge

size	GOSSIP PUSH = 0x00 0x53
challenge	
Nonce	
nodes (comma concatenated string)	

Gossip Message:

- Contains a gossip message that should be spread to all nodes in the local view when received from a known peer
- TTL should be decreased every time the message is forwarded, a message with a TTL of 1 is not forwarded further

size		GOSSIP MESSAGE
TTL	reserved	data type
data		

Security Measures

As seen in the data blocks above, we include a gossip push challenge to make it more difficult for a group of nodes to push-flood another peer.
This challenge is generated using the other peers network address.
Additionally only a certain amount of pushes from each address is allowed in each Brahms Gossip cycle.
This limits the amount of pushes an attacker can send with respect to available compute and network addresses.
Additionally the parsing is done to deal with malformatted messages safely by only receiving and parsing an expected, limited amount of bytes.

Future Work

Gossip implementation is currently being finalized, and it will adhere to the previously mentioned specifications.
Additionally, we would like to adjust the size of the view every round by integrating with the Network Size Estimator.

Workload Distribution

Together:

- We collectively wrote the reports, designed the protocol and came up with a design for the gossip implementation specification.
- Reviewing each other's work in the form of Merge Requests

Yannis:

- API Server (Packet parsing, Client registration, etc.)
- Small Config module providing parsed configuration values and defaults
- Brahms Sampler and SamplerGroup for historical view samples
- Challenge module for generation, solving and validation of PoW challenges
- GitLab CI/CD setup for compilation and container creation

Dominik:

- Brahms View
- Gossip rounds to update the view with received Pushes and requested Pulls (WIP)
- P2P API implementation as previously outlined.
- The rest is currently W.I.P.

Effort spent for the Project

In total we spent a combined amount of roughly 10 working days on the project. Currently 7 of those fall on Yannis, while 3 fall on Dominik. We expect this to even out in the near future, since Dominik currently has a lot of his work in progress that depended on some implementations done by Yannis.