Profiled Swarm

Detailed Architecture Documentation

This document outlines the core components, inter-module communication, and architectural flow of the highly customizable **Profiled Swarm** traffic generation framework, based on an analysis of the primary *Python* modules. The design prioritizes modularity and realistic simulation of complex, targeted network activities.

Architecture Overview

The system follows a typical **Manager-Worker pattern**, implementing a resilient, distributed approach to network simulation. A central Manager coordinates multiple Generators (the workers), which often operate as isolated processes or threads. The behavior of each Generator is rigorously governed by a pre-defined Profile, which dictates the statistical timing and sequencing of packet construction handled by the underlying Packets module. This structure ensures that a single configuration can launch dozens or hundreds of independent, yet coordinated, traffic streams, forming the "swarm."

Core Modules and Responsibilities

The framework is composed of seven primary *Python* modules, each with a distinct and specialized responsibility:

File	Component	Primary Role	Inter-Module Dependency
manager.py	**Manager**	Orchestrates the entire traffic generation campaign.	Depends on config.py, launches instances of generator.py.
generator.py	**Generator**	Executes the traffic generation loop according to a single profile.	Depends on profile.py and calls packets.py via the Profile.
profile.py	**Profile**	Defines the stateful, time-based behavioral models.	Depends on packets.py for template construction methods.
packets.py	**Packets**	Handles low-level network frame creation and payload crafting.	Utilizes underlying network libraries (e.g., *Scapy*[1]).
config.py	**Config**	Manages loading, validation, and hierarchical merging of settings.	Used by manager.py to bootstrap the application.
lib.py, utils.py	**Utilities**	Provides common cryptographic, time-series, and logging helpers.	Shared dependencies for all core components.
log.py	**Logger**	Standardized, thread-safe logging interface for detailed activity tracking.	Used globally for outputting execution and event data.

Manager Component (*manager.py*)

[1] https://scapy.net

The Manager is the command and control center of the Profiled Swarm, tasked with setting up the environment and deploying the workers.

- **Initialization:** reads the main configuration file (*manager.toml*) via the *config.py* module. It initializes the logging system (*log.py*) and, crucially, establishes the state persistence layer. The mention of Firebase suggests the capability to store experiment metadata, track ongoing swarm statistics in real-time, or even coordinate the swarm across multiple physical machines using a shared database.
- **Profile Instantiation & Deployment:** iterates over the defined profiles in the manager configuration. For each entry, it gathers the target parameters (host, rate limit, duration) and, using *Python*'s multiprocessing capabilities, creates and launches a new Generator instance. This parallel deployment is critical for achieving high traffic volumes and realistic concurrency.
- **Swarm Coordination and Health Monitoring:** monitors the status of all active Generator workers (the "swarm"). It handles lifecycle management (starting, stopping, restarting workers upon failure) and respects experiment-wide requirements defined in the TOML file, such as global run-time limits or maximum concurrent packet rates.

Generator Component (*generator.py*)

The Generator is the dedicated worker responsible for turning a behavioral model into live network traffic.

- **Profile Loading and Context:** receives and encapsulates a specific, fully-configured Profile object from the manager. It uses this profile as its exclusive source of truth for its current task.
- **Traffic Generation Loop:** enters a tight, primary execution loop. In each iteration, it continuously consults the encapsulated Profile to obtain the next required action and the calculated time delay until that action should occur. This loop ensures the traffic adheres to the stochastic and sequential demands of the profile.
- **Packet Sending and Rate Control:** uses the network interface specified in the configuration to transmit the constructed packets. The generator is responsible for precise timing and enforcing both the profile's internal, stochastic Inter-Packet Delay (IPD) and the strict **rate limiting** (*rate_limit_pps*) defined in the manager configuration. This two-tier control prevents any single generator from flooding the network

uncontrollably.

3. Profile Component (*profile.py*)

The Profile defines the complex behavioral model for a single traffic stream, acting as a state machine that dictates the generator's actions.

1. **Stochastic Behavioral Models:** Stores parameters defining the crucial elements of realistic traffic:
    - **Inter-Packet Arrival Times (IPTs):** uses statistical distributions (e.g., exponential for burstiness, Pareto for heavy tails, or normal for structured delays) to model the time gaps between transmissions, mimicking human *"think time."*
    - **Action Sequences:** defines the order of application-layer events (e.g., DNS Query $\rightarrow$ TCP Handshake $\rightarrow$ HTTP GET Request $\rightarrow$ Wait 5s $\rightarrow$ HTTP POST).
    - **Payload Properties:** specifies distributions for packet and payload sizes, ensuring not all generated packets are uniformly sized (which is a common giveaway of synthetic traffic).
2. **State Management:** critically, it tracks the internal state of the simulated session. This includes maintaining the current sequence number for TCP/UDP sessions, tracking cookies or session IDs for HTTP, or monitoring the progress through a predefined multi-stage attack or user journey. This ensures the traffic is **stateful** and network devices cannot simply discard the traffic based on missing session context.
3. **Decision Logic:** provides methods like *profile.get_next_action()* and *profile.get_next_delay()*. These are the interfaces the Generator uses to determine:
    a. what type of packet to request from the Packets module, and
    b. how long to sleep before executing the transmission.

4. Packets Component (*packets.py*)

This module is the framework's bridge to the network layer, translating high-level profile requests into byte-level frames.

- **Protocol Abstraction (*Scapy*/Low-Level):** it relies on a powerful low-level library (likely *Scapy* in a *Python* context) to construct headers across multiple layers: Ethernet (L2), IP (L3), and TCP/UDP (L4). This enables the generation of packets that conform to standards, avoiding the typical errors associated with hand-crafted traffic.

- **Payload Customization and Obfuscation:** includes complex functions for injecting realistic or junk data into the packet payload. This is essential for evading trivial detection by IDS systems that rely on basic pattern matching. Payloads can be sourced from randomized sequences, pre-defined samples, or even cryptographic noise.
- **Packet Templates:** defines an extensive library of pre-built, parameterized functions for common traffic types that a Profile can call: e.g., *create_http_get(url, headers)*, *create_dns_query(domain)*, *create_icmp_flood()*, or *create_ssh_session_start()*. This abstraction makes profile definition cleaner.

5. Configuration Component (*config.py* and *manager.toml*)

The configuration system handles the resilient and hierarchical loading of all operational settings.

- **manager.toml (Global Control):** the top-level configuration file that defines the orchestration parameters. It dictates the global network interface, the logging level, and most importantly, defines an array of **profile execution requests**, each pointing to a specific profile file and its target.
- **Profile-Specific Configs (Behavioral Detail):** each profile often has its own separate configuration file (e.g., a custom .toml, .json, or even a domain-specific language file). These files contain the granular details of the behavioral parameters: the exact probability weights for state transitions, the parameters (mean, σ) for statistical delay distributions, and the payload content pointers. This separation allows for highly complex behaviors to be defined and rapidly swapped without altering the swarm's core deployment strategy.

List Of Abbreviations

Abbreviations	Meaning
IPD	*Inter-Packet Delay*
DNS	*Domain Name System*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
HTTP	*HyperText Transfer Protocol*
IPT	*Inter-Packet Arrival Time*