

Project 3 - MatrixNet: The Operator's Console

CmpE 250, Data Structures and Algorithms, Fall 2025

SAs: Can Erer & Serdar Şen

TA: İrem Urhan

Instructor: Atay Özgövde

Due: December 13th, 23:59:00 Strict

1 Introduction

"Welcome to the real world."

The Matrix is everywhere. After years of covert operations, the Resistance has regained partial control of the hidden backdoor network, but the Agents have corrupted many of the old routes. The Nebuchadnezzar's crew can no longer move operatives safely using the same paths. It is time to design a new clandestine overlay before the Agents locate the hubs.

You have been appointed as the Chief Architect and Operator of **MatrixNet**, a simulation engine for designing, analyzing, and hardening the Resistance's secret tunnels inside the Matrix. Think of hosts as safe access points (phone booths, operator consoles), backdoors as the tunnels between them, and firewall ratings as layers of Agent detection. Your algorithms will decide who can move, where, and how fast, and whether Zion survives another dawn. Good luck, Operator.

Your mission (do not fail):

- Establish and remove digital access points such as ZION_CORE, TRINITY_KERNEL and MORPHEUS_HUB, and maintain the hidden backdoor tunnels between them.
- Compute optimal covert routes that respect throughput and firewall constraints so operatives like Neo and Trinity can travel unseen.
- Detect compromised tunnels and critical vulnerabilities that would fragment the infrastructure, and report them to the Oracle.
- Generate comprehensive topology reports that reveal the infrastructure's shape and weaknesses and mark critical access points and tunnels for Operator review.



Figure 1: *"There is no spoon."* Trinity uses a phone booth to exit the Matrix while Agents close in.

In this battlefield of code, performance is stealth, correctness is survival, and elegance is your signature. The Agents watch every packet, so design the infrastructure so small groups can traverse unseen.

2 Application’s Capabilities

MatrixNet enables the Resistance to create hosts and tunnels, optimize routing with bandwidth and security constraints, and analyze system topology for vulnerabilities.

2.1 How Should It Work

- The application should be able to create hosts with unique IDs and clearance levels.
- Hosts should be able to connect to other hosts through backdoor tunnels with specified latency, bandwidth, and firewall parameters.
- The system should find optimal routes between hosts while respecting bandwidth, security, and latency constraints.
- Analyze infrastructure connectivity and detect critical failure points.
- Generate comprehensive topology reports showing system statistics.

2.2 Some Warnings

- There cannot be multiple hosts with same IDs.
- Host IDs may **only** contain uppercase letters (A–Z), digits (0–9), and underscores.
- All latency and bandwidth values are positive integers. Firewall levels and clearance levels are integers in $\{1, 2, 3, 4, 5\}$.
- When multiple optimal routes exist, choose the lexicographically smallest host sequence (case-sensitive ASCII).
- Infrastructure analysis operations should work efficiently on large systems.
- You should handle cases where host IDs or tunnels don’t exist.
- All distance calculations use the appropriate metric for each command type (see below).

2.3 Global Conventions

- **Lexicographic order:** compare host IDs by case-sensitive ASCII. For sequences, compare elementwise. For an endpoint pair (a, b) with $a < b$, compare by (a, b) .
- **Rounding:** arithmetic means in reports are rounded *half up* to 1 decimal place.
- **Units and spacing:** print numbers immediately followed by their units with *no space* (e.g., 15ms, 1000Mbps, 42st).
- **Input validity:** command formats and parameter domains are guaranteed correct; still perform semantic/state validation.
- **Congestion factor:** λ in `trace_route` is a non-negative integer ($\lambda \in \mathbb{N}_0$).
- **Terminology:** "backdoor", "link", "connection", and "tunnel" are synonymous; the underlying object is a bidirectional pathway.
- **Rounding implementation:** Use `BigDecimal` with `RoundingMode.HALF_UP` to 1 decimal place.

2.4 Some Terminology

- H is the set of *hosts* (access points).
- $T \subseteq \{\{u, v\} : u, v \in H, u \neq v\}$ represents the set of *backdoor tunnels* (hidden connections) between hosts.
- For each tunnel $t = \{u, v\}$, $\ell_{\text{base}}(u, v)$ is its base latency (ms), $\beta(u, v)$ its bandwidth capacity (Mbps), and $\varphi(u, v)$ its firewall security level (in $\{1, \dots, 5\}$).
- Each host $v \in H$ carries a *clearance level* $c(v) \in \{1, 2, 3, 4, 5\}$.

3 Console Input/Output Files

All missions are executed through a single instruction file — a stream of console commands fed into the MatrixNet core. Your program must read this file, execute each instruction in order, and write a log of every event (success or error) exactly as specified.

The input will be syntactically correct and numeric parameters will respect the domains stated in this document; you do not need to validate *formats or numeric ranges*. Your job is to interpret them and simulate the infrastructure; perform semantic/state checks (e.g., existence, duplicates, already sealed, etc.).

Example Console Script:

```
spawn_host ZION_CORE 3
spawn_host TRINITY_KERNEL 2
link_backdoor ZION_CORE TRINITY_KERNEL 10 1000 2
trace_route ZION_CORE TRINITY_KERNEL 800 2
oracle_report
```

Example Output:

```
Spawned host ZION_CORE with clearance level 3.
Spawned host TRINITY_KERNEL with clearance level 2.
Linked ZION_CORE <-> TRINITY_KERNEL with latency 10ms, bandwidth 1000Mbps, firewall 2.
Optimal route ZION_CORE -> TRINITY_KERNEL: ZION_CORE -> TRINITY_KERNEL (Latency = 10ms)
--- Resistance Network Report ---
Total Hosts: 2
Total Unsealed Backdoors: 1
Network Connectivity: Connected
Connected Components: 1
Contains Cycles: No
Average Bandwidth: 1000Mbps
Average Clearance Level: 2.5
```

The output should be a chronological log showing what happened at each step (created host, backdoor linked, route found, errors, etc.).

Additional input-output test files will be provided for you to test your implementation. Time constraints for each test case type are specified in Section 6.4.

4 Console Instructions

4.1 Spawning a Host

A new *host* (safe access point) can be spawned inside MatrixNet with a unique ID and a clearance level.

Command: `spawn_host <hostId> <clearanceLevel>`

Example: `spawn_host ZION_CORE 3`

If a host with the given ID already exists, you must not overwrite it. Log:

Log on Error: Some error occurred in `spawn_host`.

When the operation succeeds, log:

Log on Success: Spawned host `<hostId>` with clearance level `<level>`.

Example Log: Spawned host `ZION_CORE` with clearance level `3`.

4.2 Linking Backdoors

A bidirectional hidden tunnel (backdoor) can be linked between two existing hosts with specified parameters.

Command: `link_backdoor <hostId1> <hostId2> <latency> <bandwidth> <firewall_level>`

Example: `link_backdoor ZION_CORE TRINITY_KERNEL 15 1000 2`

If either host does not exist, if both IDs are the same (hence self-links are prevented), or if a backdoor already exists between them, log:

Log on Error: Some error occurred in `link_backdoor`.

If the operation succeeds, log:

Log on Success: Linked `<hostId1>` <-> `<hostId2>` with latency `<latency>`ms, bandwidth `<bandwidth>`Mbps, firewall `<level>`.

Example Log: Linked `ZION_CORE` <-> `TRINITY_KERNEL` with latency `15`ms, bandwidth `1000`Mbps, firewall `2`.

4.3 Sealing or Unsealing a Backdoor

A hidden tunnel between two hosts can be toggled to simulate an Agent cutting or restoring access. Issuing the same command twice on the same backdoor flips its status (sealed ↔ unsealed).

Command: `seal_backdoor <hostId1> <hostId2>`

Example: `seal_backdoor ZION_CORE TRINITY_KERNEL`

If either host does not exist or no backdoor exists between them, log:

Log on Error: Some error occurred in `seal_backdoor`.

If the backdoor is currently *unsealed*, this command seals it:

Log on Success (Seal): Backdoor `<hostId1>` <-> `<hostId2>` sealed.

If the backdoor is already *sealed*, this command unseals it:

Log on Success (Unseal): Backdoor `<hostId1>` <-> `<hostId2>` unsealed.

Example Logs:

Backdoor `ZION_CORE` <-> `TRINITY_KERNEL` sealed.

Backdoor `ZION_CORE` <-> `TRINITY_KERNEL` unsealed.

4.4 Tracing a Covert Route

You can instruct MatrixNet to trace the optimal covert route for an operative between two hosts while respecting throughput and firewall constraints. Only unsealed backdoors whose bandwidth meets or exceeds the specified minimum may be traversed. The system uses multi-objective optimization.

Command: `trace_route <sourceId> <destId> <min_bandwidth> <lambda>`

Example: `trace_route ZION_CORE TRINITY_KERNEL 800 2`

A route $R = \langle v_1, \dots, v_k \rangle$ from source s to destination t ($v_1 = s$, $v_k = t$) is valid if

1. Each consecutive pair (v_i, v_{i+1}) corresponds to an operational (unsealed) tunnel in T for all $i = 1, \dots, k-1$;
2. $\beta(v_i, v_{i+1}) \geq \text{min_bandwidth}$ for all segments on R ;
3. **per-hop security clearance:** for each step $i = 1, \dots, k-1$, the current host must satisfy $c(v_i) \geq \varphi(v_i, v_{i+1})$.

If either host does not exist, log:

Log on Error: Some error occurred in `trace_route`.

If `<sourceId> = <destId>` and the host exists, log:

Optimal route `<id> -> <id>`: `<id>` (Latency = 0ms)

If the source and destination are the same AND the host doesn't exist, log the error message (not the self-route success message).

If no valid covert route exists with the given constraints, log:

Log on Failure: No route found from `<sourceId>` to `<destId>`

If the operation succeeds, log:

Log on Success: Optimal route `<sourceId> -> <destId>`: `<host1> -> <host2> -> ... -> <destId>` (Latency = `<totalLatency>`ms)

Example Log: Optimal route `ZION_CORE -> TRINITY_KERNEL`: `ZION_CORE -> MORPHEUS_HUB -> TRINITY_KERNEL` (Latency = 37ms)

Operator's Note — Dynamic Latency and Congestion Factor.

A covert route is valid if each backdoor along it is unsealed, meets the minimum bandwidth requirement, and at each step i the current host's clearance satisfies $c(v_i) \geq \varphi(v_i, v_{i+1})$ (per-hop clearance).

Note: Only the clearance at the source host of each tunnel segment is checked; the destination host requires no clearance check.

The parameter λ in the command is the congestion factor and is a non-negative integer: it models how much extra latency each successive step adds to the route.

For a route $R = \langle v_1, \dots, v_k \rangle$ with tunnel segments (v_i, v_{i+1}) , the effective (dynamic) latency of segment (v_i, v_{i+1}) is:

$$\ell_{\text{eff}}(v_i, v_{i+1}) = \ell_{\text{base}}(v_i, v_{i+1}) + \lambda \cdot (i - 1)$$

where i is the segment's position in the route (**step index; 1 for the first segment, 2 for the second, ...**).

Compare candidate routes using the following order:

- (1) smallest total dynamic latency (sum of all ℓ_{eff})
- (2) then fewer segments,
- (3) then lexicographically smaller full host sequence.

4.5 Scanning the Grid

You can instruct MatrixNet to scan the grid and determine whether the Resistance infrastructure is fully connected through currently unsealed backdoors.

Command: `scan_connectivity`

By convention, if $|H| \in \{0, 1\}$, the infrastructure is considered fully connected.

If the infrastructure is fully connected, log:

Log on Success: Network is fully connected.

If the infrastructure is disconnected, log:

Log on Failure: Network has <num> disconnected components.

4.6 Simulating a Breach

This command performs a temporary breach analysis without permanently modifying the Resistance infrastructure. It supports two forms:

(A) Host Breach Temporarily remove a single host and report whether it is an articulation point and, if so, how the infrastructure fragments.

Command: (host breach) `simulate_breach <hostId>`

Example: `simulate_breach TRINITY_KERNEL`

If the host does not exist, log:

Log on Error: Some error occurred in `simulate_breach`.

Otherwise, temporarily remove the host and all its incident backdoors, then analyze connectivity using only unsealed tunnels among the remaining hosts. You must log one of:

Host <hostId> is NOT an articulation point. Network remains the same.

or

Host <hostId> IS an articulation point.
Failure results in <k> disconnected components.

(B) Backdoor Breach Temporarily remove a single backdoor and report whether it is a bridge and, if so, how the infrastructure fragments.

Command (backdoor breach): `simulate_breach <hostId1> <hostId2>`

Example: `simulate_breach ZION_CORE TRINITY_KERNEL`

If either host does not exist or there is no backdoor between them, log:

Log on Error: Some error occurred in `simulate_breach`.

Otherwise, if the backdoor is already sealed, log:

Log on Error: Some error occurred in `simulate_breach`.

If the backdoor is unsealed, temporarily treat it as sealed and analyze connectivity using only unsealed tunnels. You must log one of the following:

Backdoor <hostId1> <-> <hostId2> is NOT a bridge. Network remains the same.

or

Backdoor <hostId1> <-> <hostId2> IS a bridge.
Failure results in <k> disconnected components.

Notes

- The analysis is temporary; after logging, restore the original infrastructure state (host/backdoor remains as before).
- All analyses must consider only currently *unsealed* backdoors.

4.7 Consulting the Oracle: Topology Report

You can instruct MatrixNet to consult the Oracle and generate a comprehensive *Topology Report* of the current backdoor infrastructure.

Command: `oracle_report`

Your log should look like:

```
--- Resistance Network Report ---
Total Hosts: <count>
Total Unsealed Backdoors: <count>
Network Connectivity: Connected/Disconnected
Connected Components: <count>
Contains Cycles: Yes/No
Average Bandwidth: <average>Mbps
Average Clearance Level: <average>
```

Example Log:

```
--- Resistance Network Report ---
Total Hosts: 6
Total Unsealed Backdoors: 8
Network Connectivity: Connected
Connected Components: 1
Contains Cycles: Yes
Average Bandwidth: 825.0Mbps
Average Clearance Level: 2.5
```

Operator's Note — Cyclic Structure.

A cyclic structure *exists* if there is a closed traversal sequence $\langle v_1, v_2, \dots, v_k, v_1 \rangle$ with $k \geq 3$ where all hosts v_1, \dots, v_k are distinct (except for first & last), using only unsealed backdoor tunnels.

5 Uploading to the Mainframe

Submit a single `.zip` file containing all of your Java source files via Moodle. Place every `.java` file in the zip and keep your main entry point named `Main.java`.

Name your zip file in this format:

Structure: `<studentnumber>.zip`

Your code must compile and run correctly with:

```
javac *.java
java Main <input_file> <output_file>
```

6 Grading by the Oracle

6.1 Types and Grading

There are 3 input types — each corresponds to a class of operations the Resistance requires:

- **Type 1: Host and Backdoor Management** — spawning hosts with clearance levels and linking/sealing backdoors.
- **Type 2: Covert Routing** — multi-objective route optimization considering dynamic latency, bandwidth and firewall constraints.
- **Type 3: Network Analysis** — connectivity scans, breach simulations and full Oracle reports.

6.2 Grade Distribution

See Table 1 for grade distribution for small and large cases:

Type	Small Case (%)	Large Case (%)
1	15	20
2.1 ($\lambda = 0$)	4	6
2.2 ($\lambda \geq 0$)	14	16
3.1 (only <code>scan_connectivity</code>)	3	4
3.2 (everything included)	12	6

Table 1: Grade Distribution for Small and Large Cases

6.3 Constraint Specifications

Constraint specifications (maximum number of commands per type) are in Table 2.

Input Type	Type 1 (S)	Type 1 (L)	Type 2 (S)	Type 2 (L)	Type 3 (S)	Type 3 (L)
<code>num_spawn_host</code>	10000	100000	5000	50000	10000	50000
<code>num_link_backdoor</code>	50000	500000	10000	100000	5000	20000
<code>num_seal_backdoor</code>	50000	500000	2000	2000	1000	10000
<code>num_trace_route</code>	0	0	300	500	50	500
<code>num_scan_connectivity</code>	0	0	0	0	50	500
<code>num_simulate_breach</code>	0	0	0	0	50	500
<code>num_oracle_report</code>	0	0	0	0	50	500

Table 2: Maximum Values of Different Types (Small and Large Versions)

- **`num_spawn_host`**: Expected number of host creation commands.
- **`num_link_backdoor`**: Expected number of backdoor link commands.
- **`num_seal_backdoor`**: Expected number of backdoor sealing commands.
- **`num_trace_route`**: Expected number of route-tracing commands.
- **`num_scan_connectivity`**: Expected number of connectivity scans.
- **`num_simulate_breach`**: Expected number of breach simulations.
- **`num_oracle_report`**: Expected number of Oracle reports.

6.4 Time Constraints

Each test case has a time constraint (seconds) according to Table 3:

Type	Small Case (seconds)	Large Case (seconds)
1	2	5
2	2	30
3	2	30

Table 3: Time Constraints for Small and Large Cases

These constraints are based on our reference machine (11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz, 24GB RAM), machine plugged in. Keep this in mind while testing your code locally.

Operator's Note — Reproducibility. All command outputs must follow the exact formats and ordering rules given in this specification. Different orders of equally valid results will be graded as incorrect. All outputs must match the exact strings including punctuation. When an ordering is specified (routes, tunnels, regions), that ordering is mandatory; any other order is graded incorrect.

7 Warnings

- **Original Work.** All submissions are automatically compared against other students' code and previous years' exercises. Write and submit your *own* implementation. Any sign of cheating will incur at least -100 points on the first offense and formal disciplinary action if repeated.
- **Readable Code.** Document your code with concise inline comments and use meaningful variable names. Avoid over-commenting or excessively long identifiers. Clear code is essential for partial grading.
- **Exact Output Format.** Your output must follow the exact text and ordering rules given in this document. Only trailing white-spaces at the end of lines may be disregarded.
- **Discussion Forum.** Use the Moodle discussion forum for your questions and always check if an answer already exists before posting.
- **No Built-In Libraries or Priority Queues.** You are **not** allowed to use built-in data structures (You can use `java.util.ArrayList` & `java.util.LinkedList`).
- **Late Submission.** The following penalties are applied for late submissions:

Days Late	Penalty (%)
≤ 1	-1
≤ 2	-3
≤ 3	-7
≤ 4	-15
≤ 5	-31
> 5	-100

Table 4: Late Submission Policy