UE18CS390B – Capstone Project Phase – 2

SEMESTER - VII

END SEMESTER ASSESSMENT

Project Title :Pseudo-System Protocol for Information Transfer
Project ID :PW22PG01
Project Guide :Prof. Pushpa G
Project Team :Akshay Vasudeva Rao
Amogh R K
Prajwal K Naik
Rohan Iyengar

Outline

- Abstract
- Team Roles and Responsibilities.
- Summary of Requirements and Design (Capstone Phase - 1)
- Summary of Methodology / Approach (Capstone Phase - 1)
- Design Description
- Modules and Implementation Details
- Project Demonstration and Walkthrough
- Test Plan and Strategy
- Results and Discussion
- Lessons Learnt
- Conclusion and Future Work
- References

Abstract

This protocol secures the data being transferred and masks the host so that a third party cannot intervene for stealing data or gain access to the hosts. The protocol sequences the transfer of data such that the hosts always remain disconnected from the internet while using the internet as the primary medium for the transfer, thus making full use of the power of the internet but without any of the risks of intrusion that come with it.

Team Roles and Responsibilities

	Akshay	Amogh	Prajwal	Rohan
Review-1	Paper 1	Paper 2	Paper 3	Paper4
Review-2	HB for Client	HB Pseudo-Client	HB for Pseudo-Server	HB for server
Review-3	Router configuration experimentation RFC	Troubleshooting for Multisystem Low level design document	Multi system - VPN Implementation Low level Design Document	VPN identification Low level design document
Review-4	Poster Conf. Identification	Report Research Paper	Report Research Paper	Poster Other Deliverables
LOC	240	250	260	240

Summary of Requirements and Design

The major functional and non-functional requirements of the protocol included:

- The standard system must have access to the internet just like a regular system using TLS to access the internet
- Data Security - The payload must not be easily decryptable.
- Host Security - The host system must not be in direct communication with the internet
- Low latency delivery - Additional levels of latency must not be unacceptable by today's standards of internet usage.
- Must follow IEEE standards for internet communication for layering over TCP as an additional layer of security.
- Pseudo-system must have no knowledge of the standard system's communication details.
- At any moment in time when the pseudo-systems are communicating via the internet, the standard client must remain in an isolated state.
- The pseudo-system must periodically accept incoming probes from the standard system

Summary of Requirements and Design

Design approach:

In the development of Pseudo-system Protocol for Information Transfer, the design approach followed was iterative, and prototypes were created as a proof of concept at every step.
From the initial prototype - a bare-bones python prototype run on a single system using the Sockets API to simulate packet flow

To a multi-system implementation, where packets move like they would in the real world, using real infrastructure for intercommunication.

In this iterative prototype-based approach, research was easier to conduct, and the approach is better suited to find flaws, and work them out through consecutive prototypes

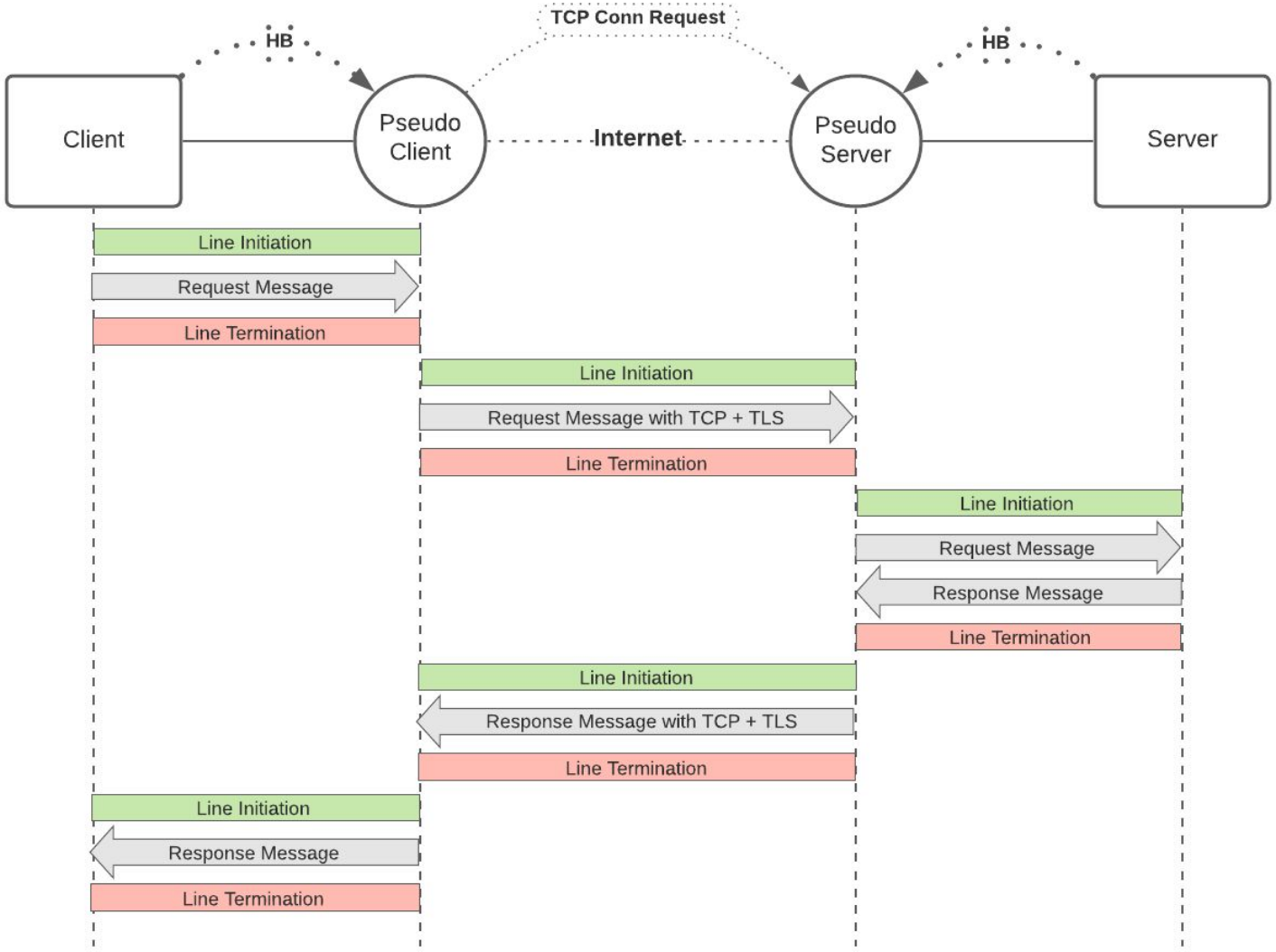
Summary of Requirements and Design

Constraints:

- The system must operate within acceptable limits of latency for the standard system.
- The pseudo-system must not be able to initiate communications with the standard system in any way, shape, or form.
- Any and all communications must be initiated by the standard system over a local area network.
- Any user opting to use PSPIT must have access to two systems communicating over a local area network in order to use it.
- Complex payloads must go through additional resolution before passing through PSPIT’s framework (video packets, teleconferencing, etc.)

Design Description

Model Architecture



Summary of Methodology / Approach

The segmented design approach of having four separate machines with their own code, running in tandem with each other makes interoperability in a real world scenario much easier.

Codebases can be developed independently, and can be machine agnostic, which reduces the complexity of code development.

The sequential design of PSPIT allows for transparency, enabling the users to be aware of any and all actions taking place during inter-system communications.

A drawback of the segmented approach would be that if one system would need a software upgrade, then all the others must be upgraded in step, for the protocol to continue working.

Summary of Methodology / Approach

Technologies used:

- Python 3.x
- Transmission Control Protocol
- Secure Socket Layer
- Transport Layer Security
- Asymmetric encryption
- Transport layer packet flow manipulation
- Sockets API

Summary of Methodology / Approach

Change in approach:

On completion of phase-1, we encountered the issue on how to combat pseudo-system intrusion, and how the standard system would be impacted in that case.

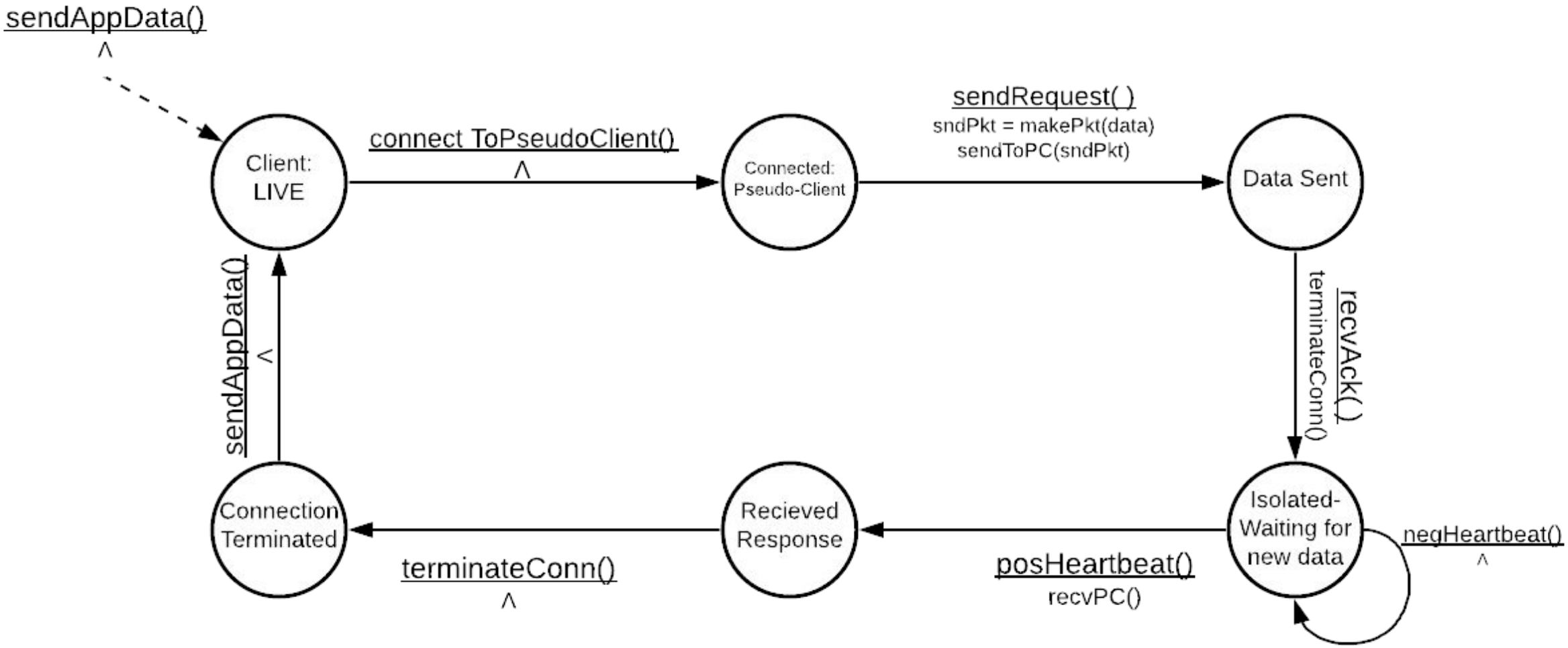
To solve this issue, we implemented the feature of “heartbeats”, which enables the pseudo-system to intermittently open its incoming ports for incoming data.

In case of no incoming packets, the pseudo-system closes the ports, and again continues intermittently accepting packets, if any.

This was a major addition to our protocol, which solved an issue that we seeded out after phase-1.

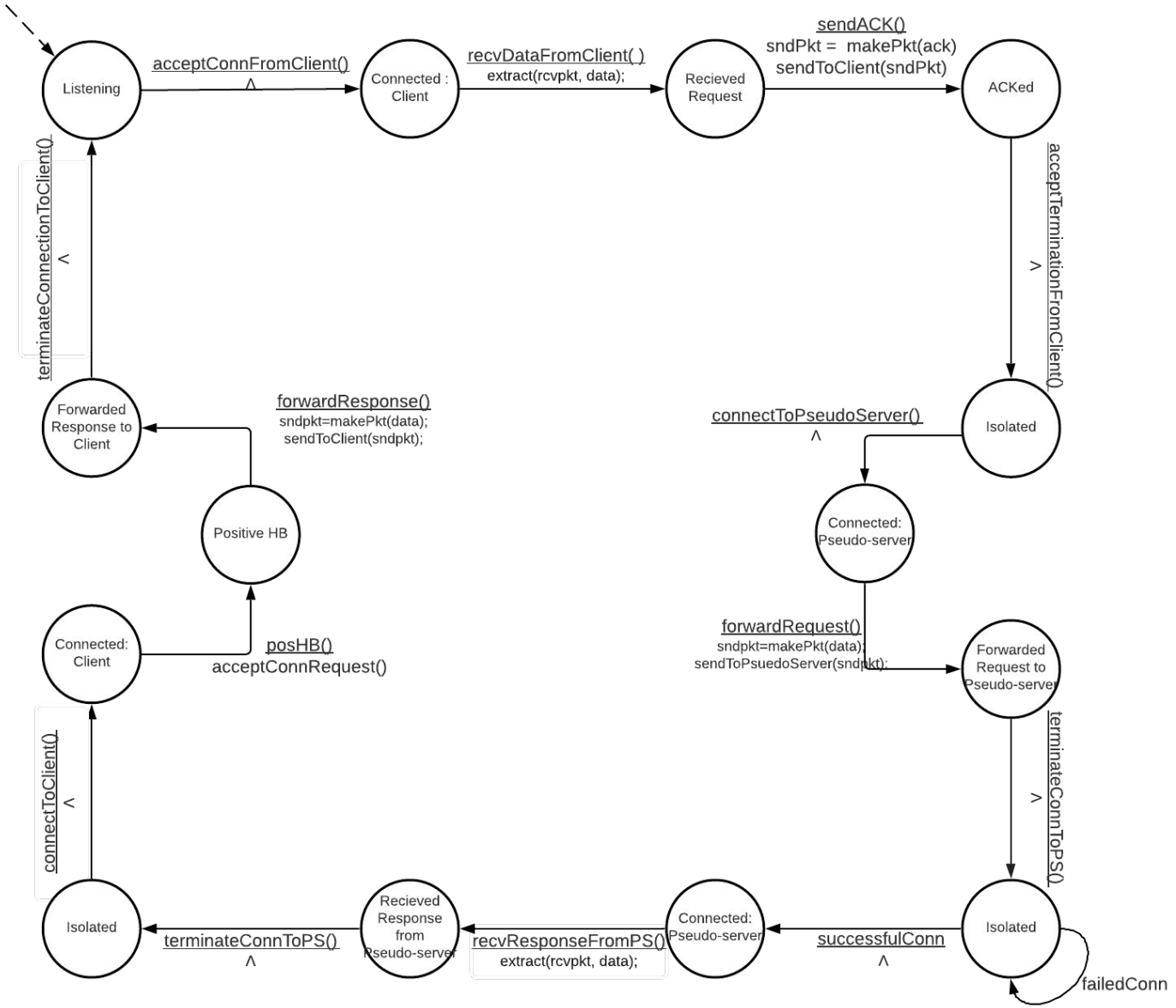
Design Description

Client Module



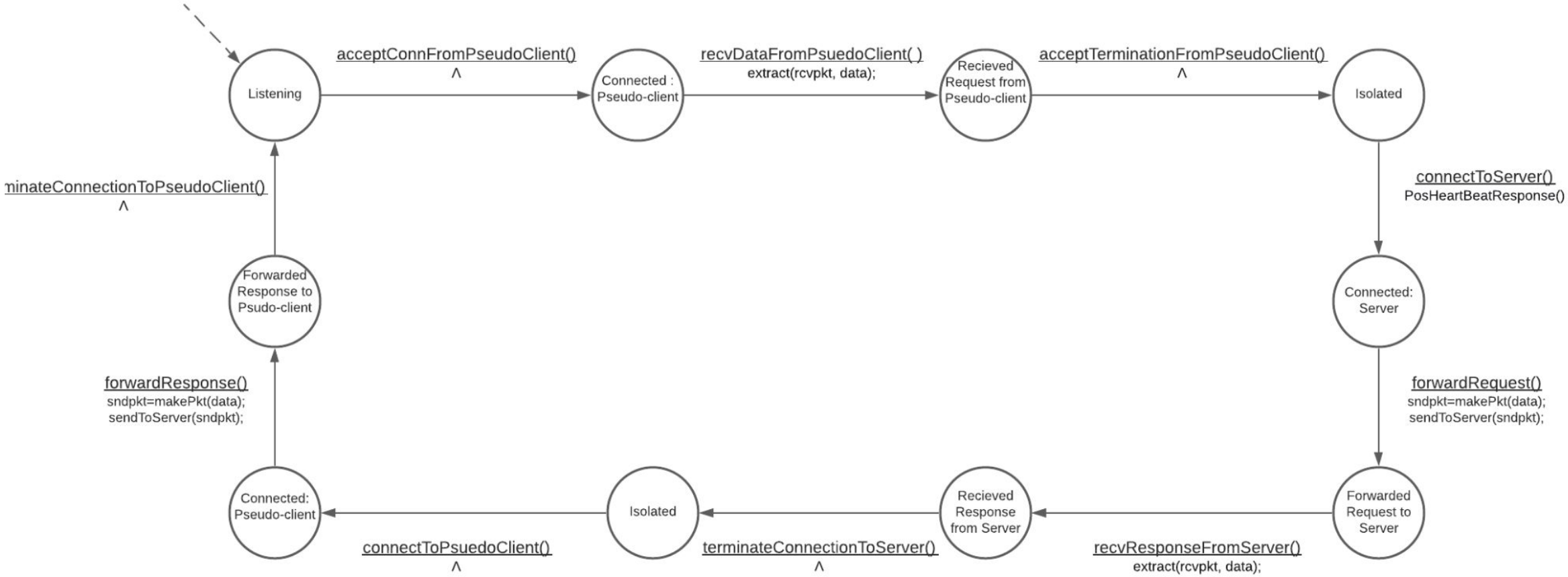
Design Description

Pseudo-Client



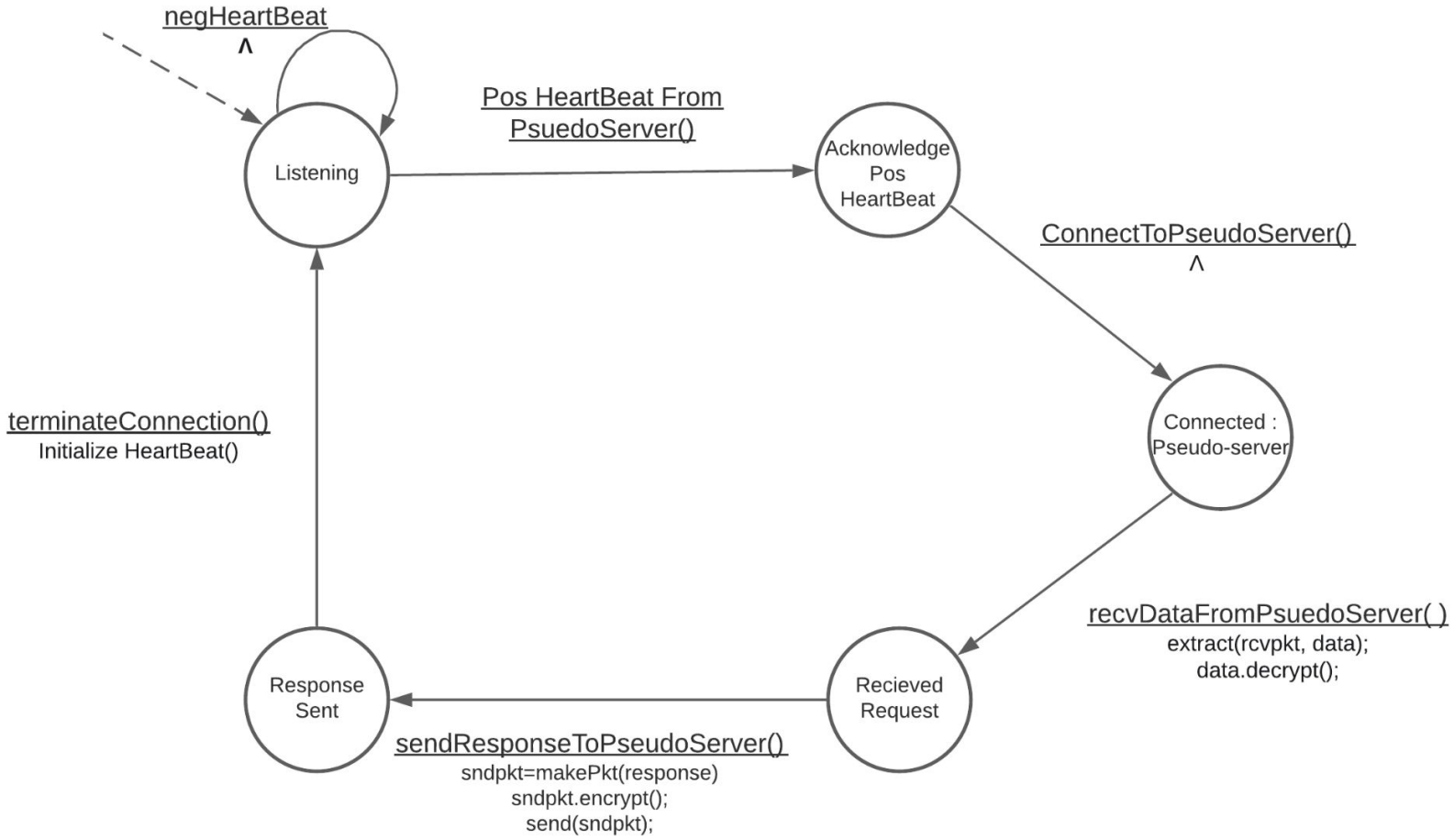
Design Description

Pseudo-Server Module



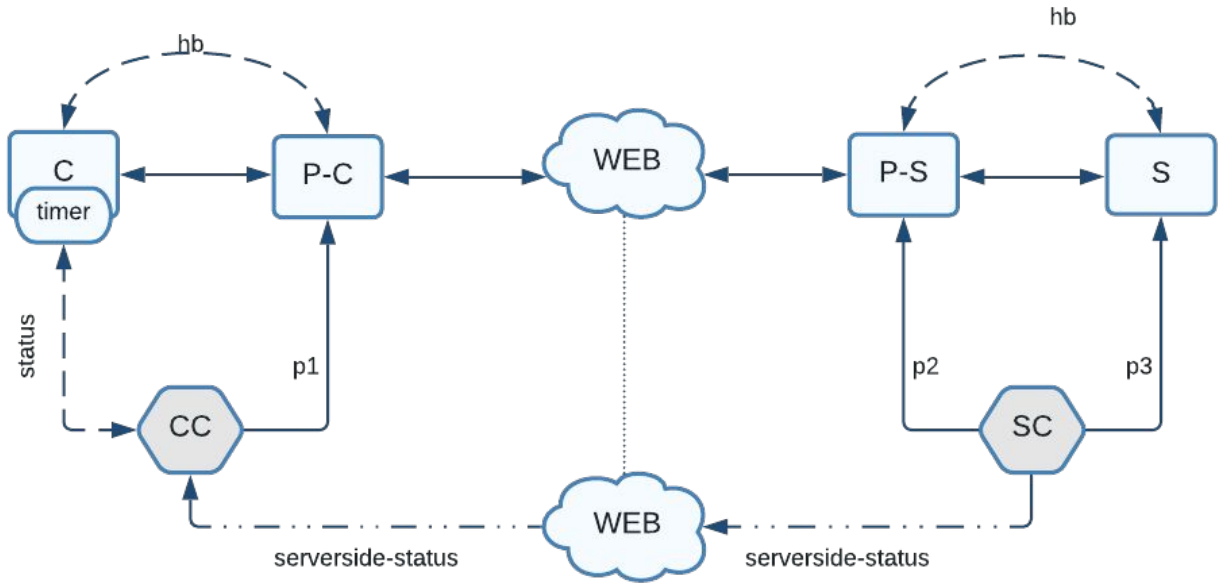
Design Description

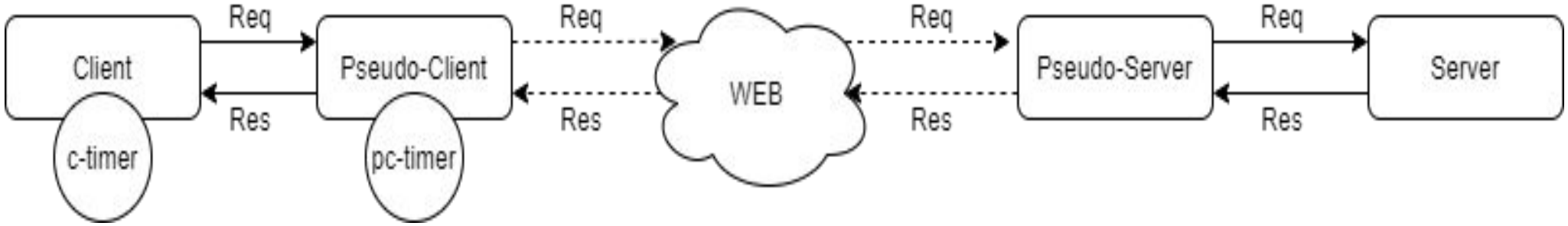
Server Module



Design Description

System Failure Handling





Modules and Implementation Details

Client Pseudo-Code

Sending Request to Pseudo-Client	clientSocket = createSocket() clientSocket.connect(pseudoClient) pkt = createReqPacket() clientSocket.sendReq(pkt) ack = recvAck() clientSocket.close()

Receive Response from Pseudo-Client	//use socket returned from heartbeat module as clientSocket
	serverResponse = clientSocket.recieve()
	clientSocket.close()

Heartbeats	while(true):
	try:
	clientSocket = createSocket(pseudoClient).connect()
	hb = createPacket(heartBeat)
	clientSocket.send(hb)
	hb_reponse = clientSocket.receive()
	if(hb = positive):
	return clientSocket
	except:
	if(clientSocket):
	clientSocket.close()

Modules and Implementation Details

Pseudo-client pseudo-code

Receiving Request from Client	fromClientSocket = createSocket(client) fromClientSocket.acceptConnection() request = fromClientSocket.receive() ack = makePacket("ACK") fromClientSocket.send(ack) fromClientSocket.close()

Receiving Response from Pseudo-Server	response = pseudoServerSocket.receive() pseudoServerSocket.close()

Modules and Implementation Details

Pseudo-client pseudo-code

Forwarding to Pseudo-Server	pseudoServerSocket = createSocket(pseudoServer)
	//data received from client
	request = makePacket(data)
	pseudoServerSocket.send(request)
	pseudoServerSocket.close()
	while(true):
	  try:
	  pseudoServerSocket = createSocket(pseudoServer)
	  break
	  except:
	  pass
	  sleep(1) //This can be a moving average value

Forwarding Response to Client	respondHeartbeat(client)
	//toClient socket obtained from respondHeartbeat module
	toClientSocket.send(response)
	toClientSocket.close()

respondHeartbeat	toClientSocket = createSocket(client) toClientSocket.connect() hb = recieveHB() toClientSocket.send("NEW") return toClientSocket

Modules and Implementation Details

Pseudo-Server Code

Receiving Request from Pseudo-Client	pseudoClientSocket = createSocket(pseudoClient) request = pseudoClientSocket.receive() ack = makePacket("ACK") pseudoClientSocket.send(ack) pseudoClientSocket.close()

Forwarding Request to Server	respondHeartBeat()
	//serverSocket returned by respondHeartBeat module
	request = makePacket(request)
	serverSocket.send(request)

Receive Response to Server	response = serverSocket.receive() serverSocket.close()

Forward Response to Pseudo-Client	pseudoClientSocket = makeSocket(pseudoClient) pseudoClientSocket.acceptConnReq() response = makePacket(response) pseudoClientSocket.send(response) pseudoClientSocket.close()
Respond Heart Beat	serverSocket = createSocket(server) hb = serverSocket.receive() hb_res = makePacket("NEW") serverSocket.send(hb_res) return serverSocket

Modules and Implementation Details

Server Code

Receive Request from Pseudo-Server	`//pseudoServerSocket received from heartbeat module` `request = pseudoServerSocket.receive()`

Respond to Request	`response = makePacket(reponse)`
	`pseudoServer.send(response)`
	`pseudoServer.close()`

Heartbeats	`while(true):`
	`  try:`
	`  pseudoServerSocket = createSocket(pseudoServer)`
	`  hb = makePacket(hb)`
	`  pseudoServerSocket.send(hb)`
	`  hb_response = pseudoServer.receive()`
	`  if(hb_reponse == "NEW"):`
	`  return pseudoServerSocket`
	`  except:`
	`  if(pseudoServerSocket):`
	`  pseudoServerSocket.close()`

Walkthrough

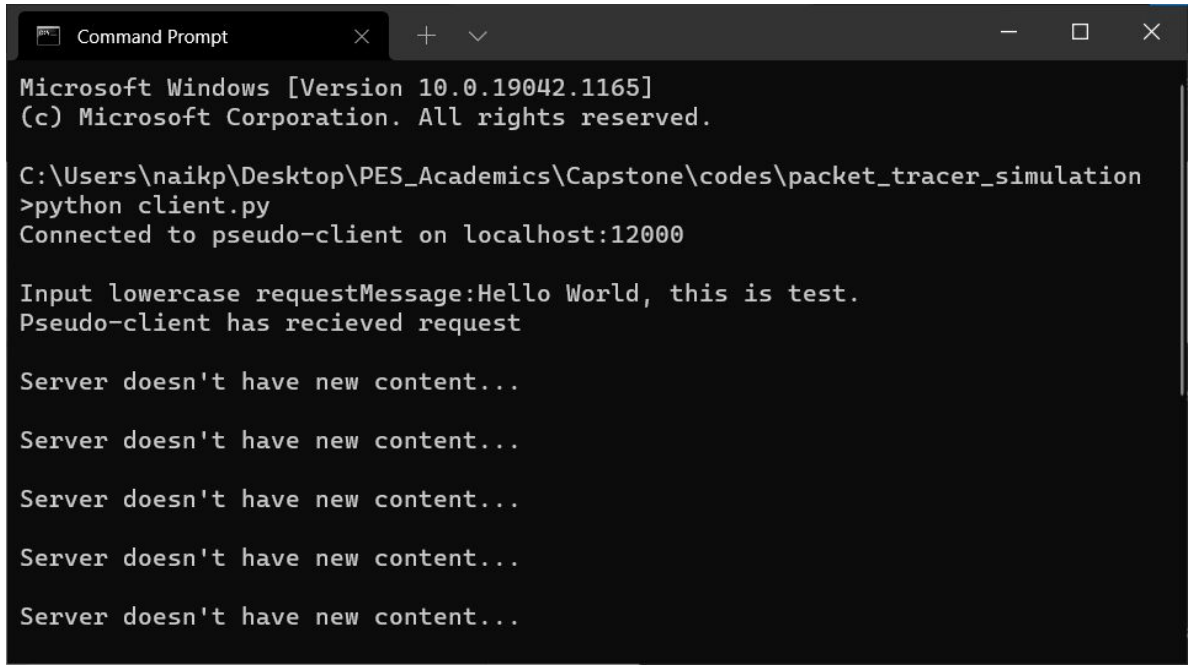
PES UNIVERSITY

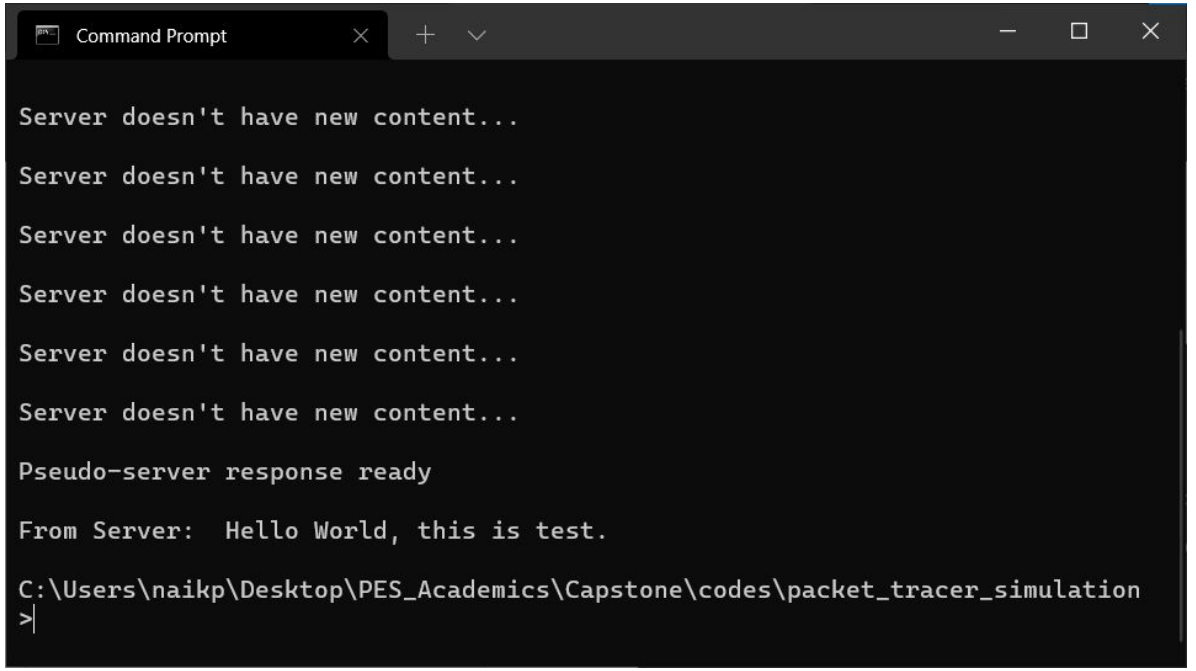
DEMO WALK THRU

Test Plan and Strategy

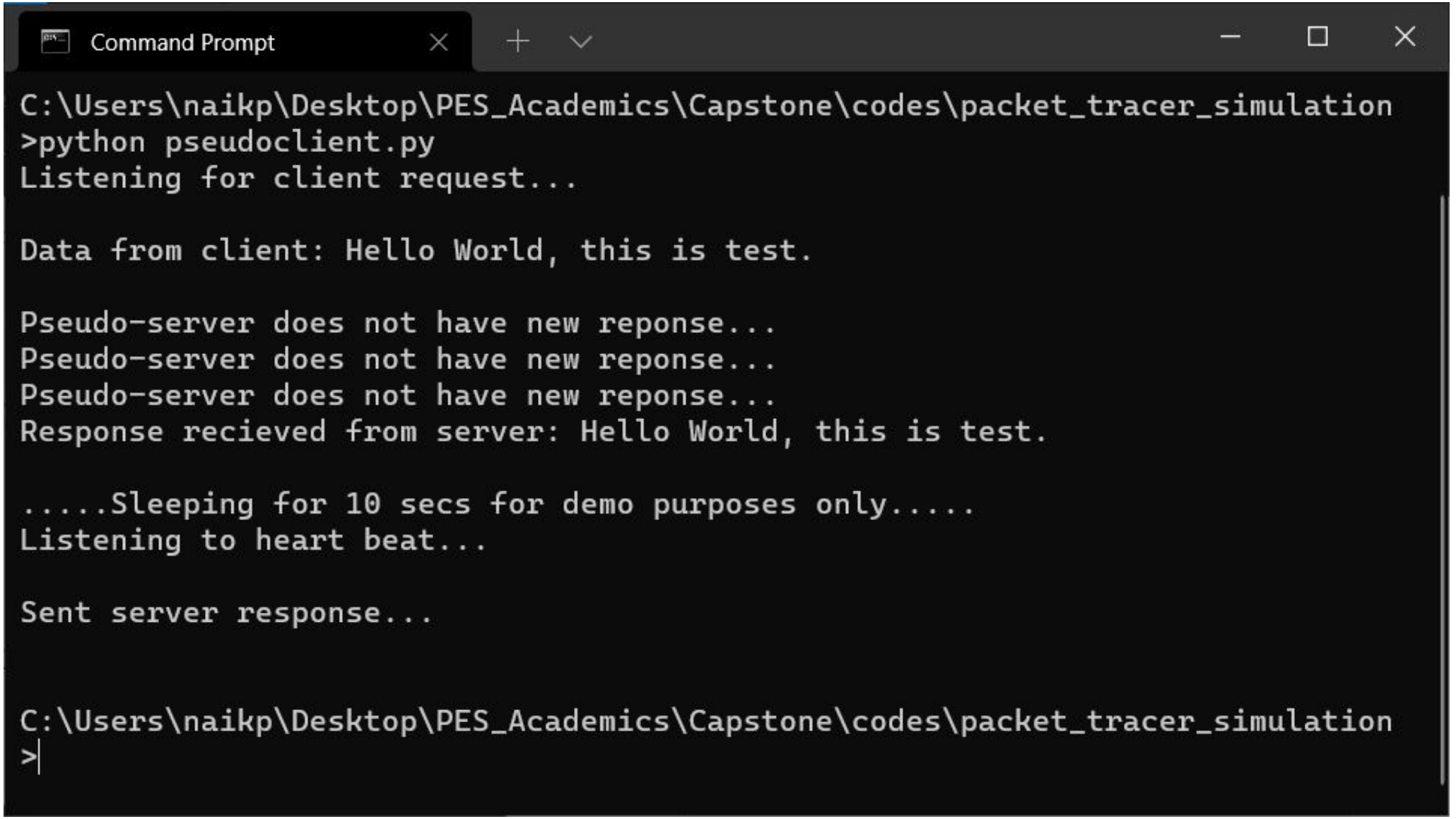
Test Case ID	Test case description	Pre-conditions	Test Steps	Test data	Expected Results	Actual Result	Test Result
UT-01	String exceeding 1024 bytes being transmitted.	None	sendData = <long_string>	A string exceeding 1024 bytes.	Truncated transmission.	Truncated transmission	Pass
UT-02	String with non ASCII characters	None	sendData = <String_of_emoji>	String of emojis	Successful transmission	Successful transmission	Pass
UT-03	Starting pseudo-server before server	None	Starting pseudo-server before server	None	Failed transmission	Failed transmission	Pass
UT-04	Starting pseudo-client before pseudo-server	None	Starting pseudo-client before pseudo-server	None	Failed transmission	Failed transmission	Pass
UT-05	Starting client before pseudo-client	None	Starting pseudo-client before pseudo-server	None	Failed transmission	Failed transmission	Pass
UT-06	Empty string being sent.	None	sendData = “”	sendData = “”	“”	Empty packet will be displayed	Pass

Results and Discussion

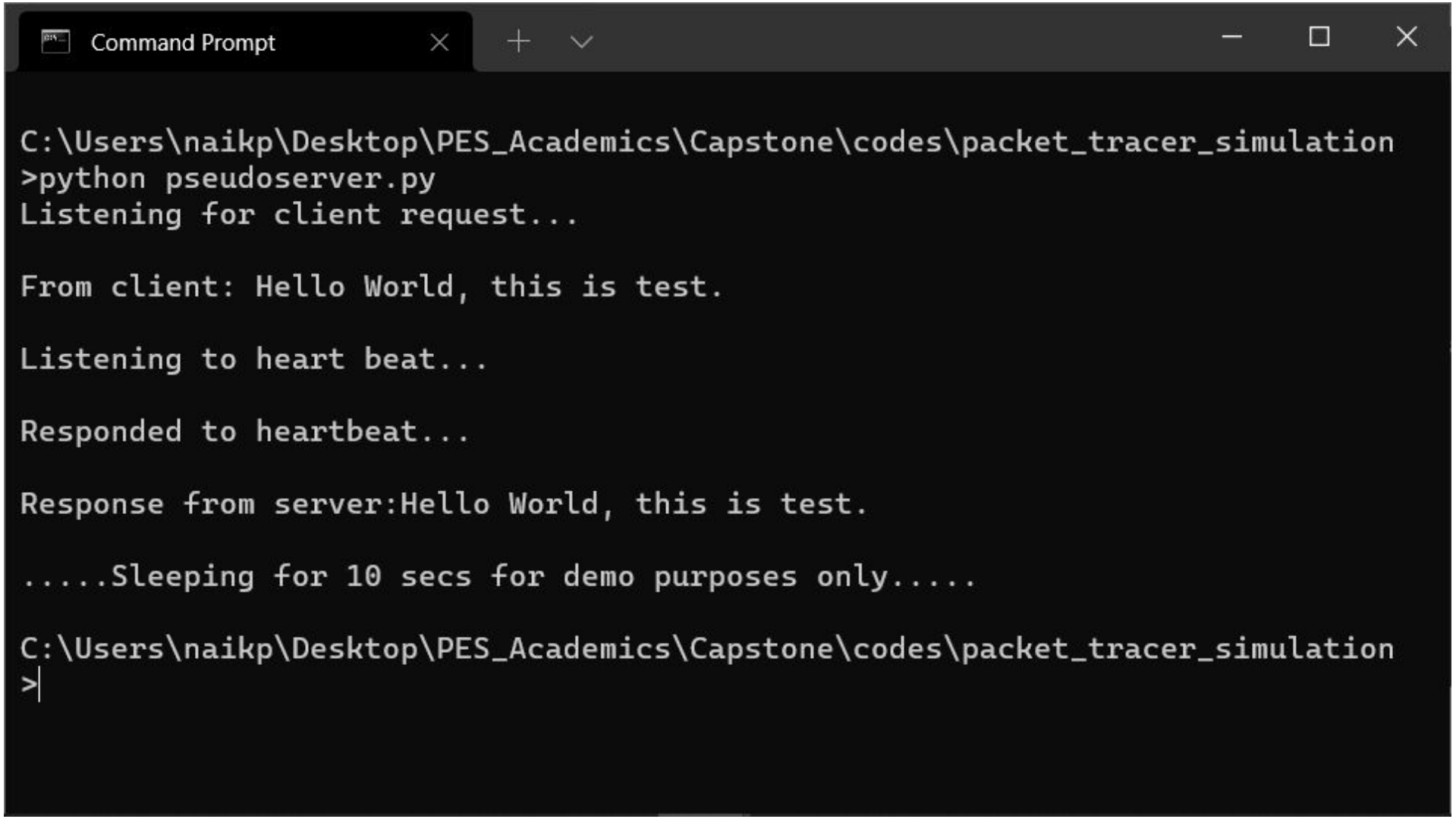




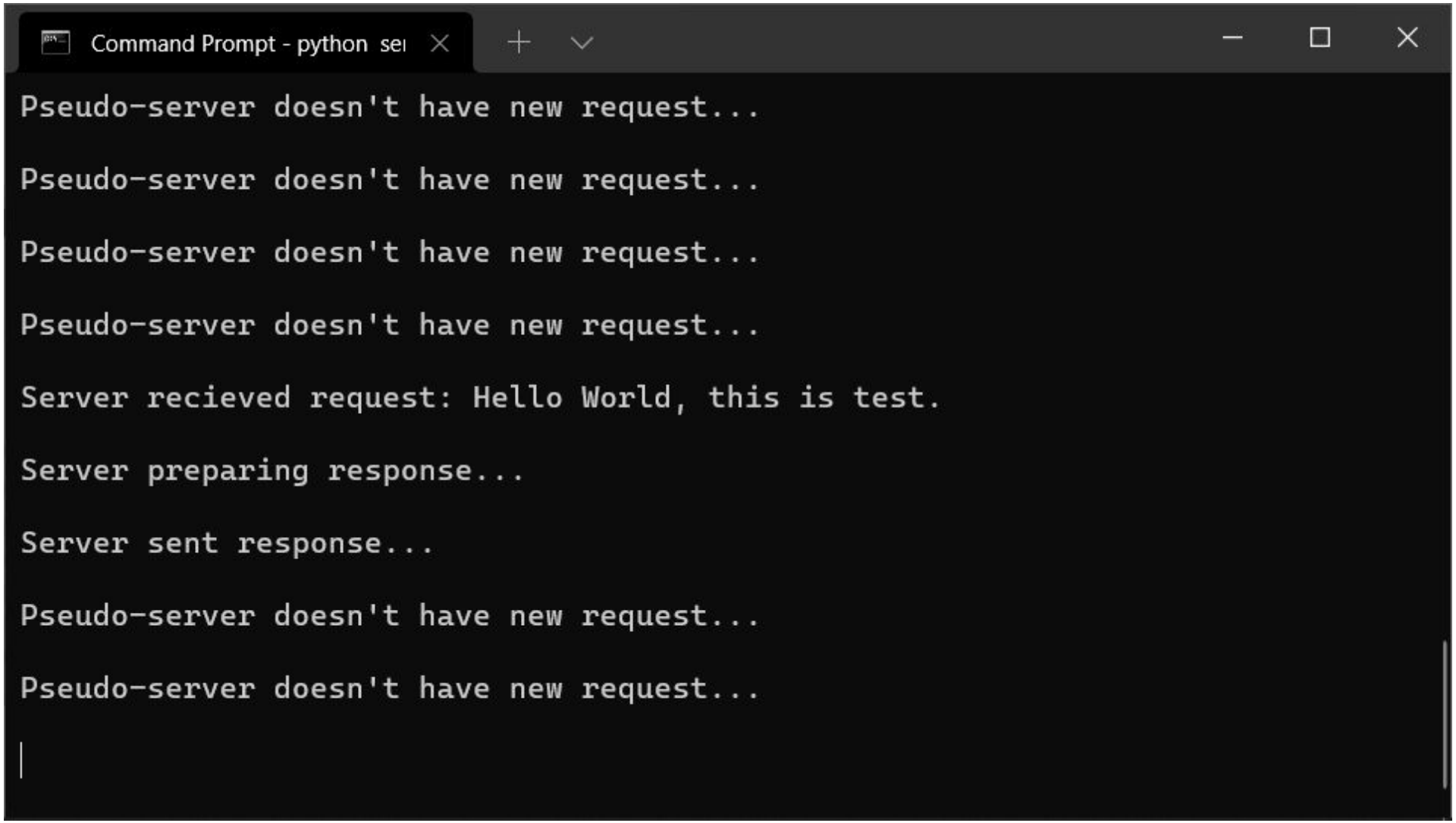
Results and Discussion



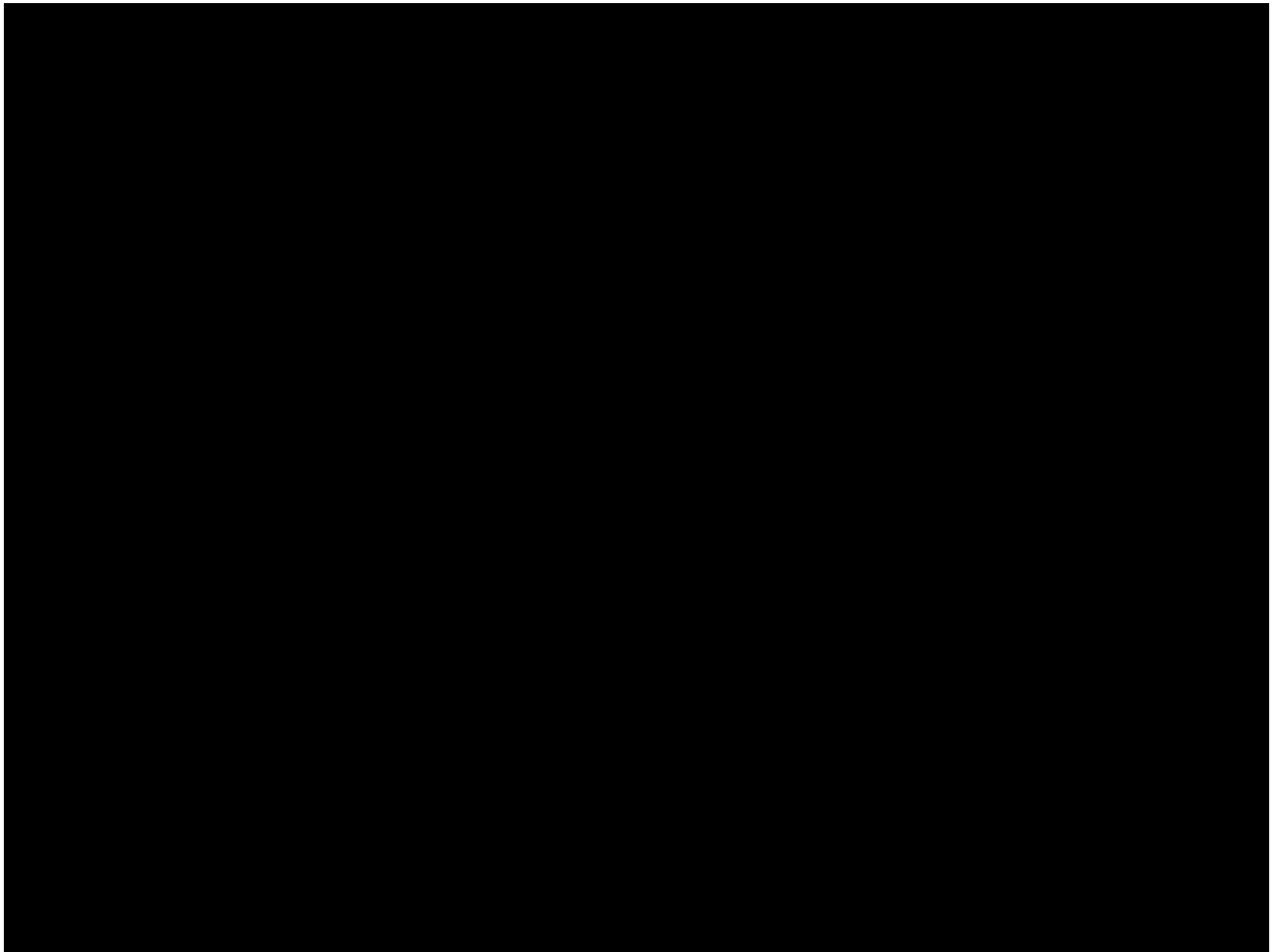
Results and Discussion



Results and Discussion



Documentation - Demo Video



Documentation

- Project report finalized by Guide.
- IEEE (similar) Format of Paper ready for submission.
- Target Conference: 3rd International Conference for Emerging Technology 2022
- GitHub Repository: https://github.com/prajwal-naik/pspit-ui-local
- Poster: https://drive.google.com/file/d/1YsqaNW3SIEHKfKypjhlYIP66hYXzJDea/view?usp=sharing
- All artifacts of project uploaded in the CSE Project repository.

Lessons Learnt

- TLS-SSL, and asymmetric encryption
- In order to make the system entirely invisible to the public internet, heartbeats/probes are the most effective solution
- Distributed computing systems have multiple points of susceptibility
- Commercial router configurations can vary massively and had to be compensated with a VPN tunnel.
- Segmented code design allows for easy interoperability, but increases lines of code, and update complexity.

Conclusion and Future work

The future work for the protocol involves working on further safeguards localized in the pseudo-systems which includes but is not limited to

- Partial execution of code
- Anti-malware screening of payload
- Stability testing of payload
- Encrypted sandbox testing

Furthermore, these multi-system communications will be encrypted with the use of TLS, which is Transport Layer Security. TLS is a secure communication encryption standard used worldwide for encrypting internet communications from unintended and/or malicious attackers. TLS uses keys in the form of certificates, cipher and decipher messages, and the certificates are confidentialy maintained by each pseudo-system. This TLS security layer acts on top of TCP, with the systems sending their certificates to each other via the protocol, following which messages are encrypted with the destination machine’s public key, and decrypted with its private key.

References

- Elsadig, Muawia & Fadlalla, Yahia. (2016). Survey on Covert Storage Channel in Computer Network Protocols: Detection and Mitigation Techniques. International Journal of Advances in Computer Networks and Its Security. 6.
- Pawar, Mohandas & Anuradha, J.. (2015). Network Security and Types of Attacks in Network. Procedia Computer Science. 48. 10.1016/j.procs.2015.04.126.
- Pandey, Shailja. (2011). MODERN NETWORK SECURITY: ISSUES AND CHALLENGES. International Journal of Engineering Science and Technology. 3.
- Manu, A. & Rudra, Bhawana & Reuther, Bernd & Vyas, O.. (2011). Design and Implementation Issues of Flexible Network Architecture. 10.1109/CICN.2011.59.
- Satapathy, Ashutosh & Livingston, Jenila. (2016). A Comprehensive Survey on SSL/ TLS and their Vulnerabilities. International Journal of Computer Applications. 153. 31-38. 10.5120/ijca2016912063.
- Sirohi, Preeti & Agarwal, Amit & Tyagi, Sapna. (2016). A comprehensive study on security attacks on SSL/TLS protocol. 893-898. 10.1109/NGCT.2016.7877537.
- Atighetchi, Michael & Soule, Nate & Pal, Partha & Loyall, Joseph & Sinclair, Asher & Grant, Robert. (2013). Safe Configuration of TLS Connections - Beyond Default Settings. 6th IEEE Symposium on Security Analytics and Automation (SafeConfig 2013).
- Castelluccia, Claude & Mykletun, Einar & Tsudik, Gene. (2006). Improving secure server performance by re-balancing SSL/TLS Handshakes. 2006. 26-34. 10.1145/1128817.1128826.
- Larisch, James & Choffnes, David & Levin, Dave & Maggs, Bruce & Mislove, Alan & Wilson, Christo. (2017). CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. 539-556. 10.1109/SP.2017.17.

Thank You