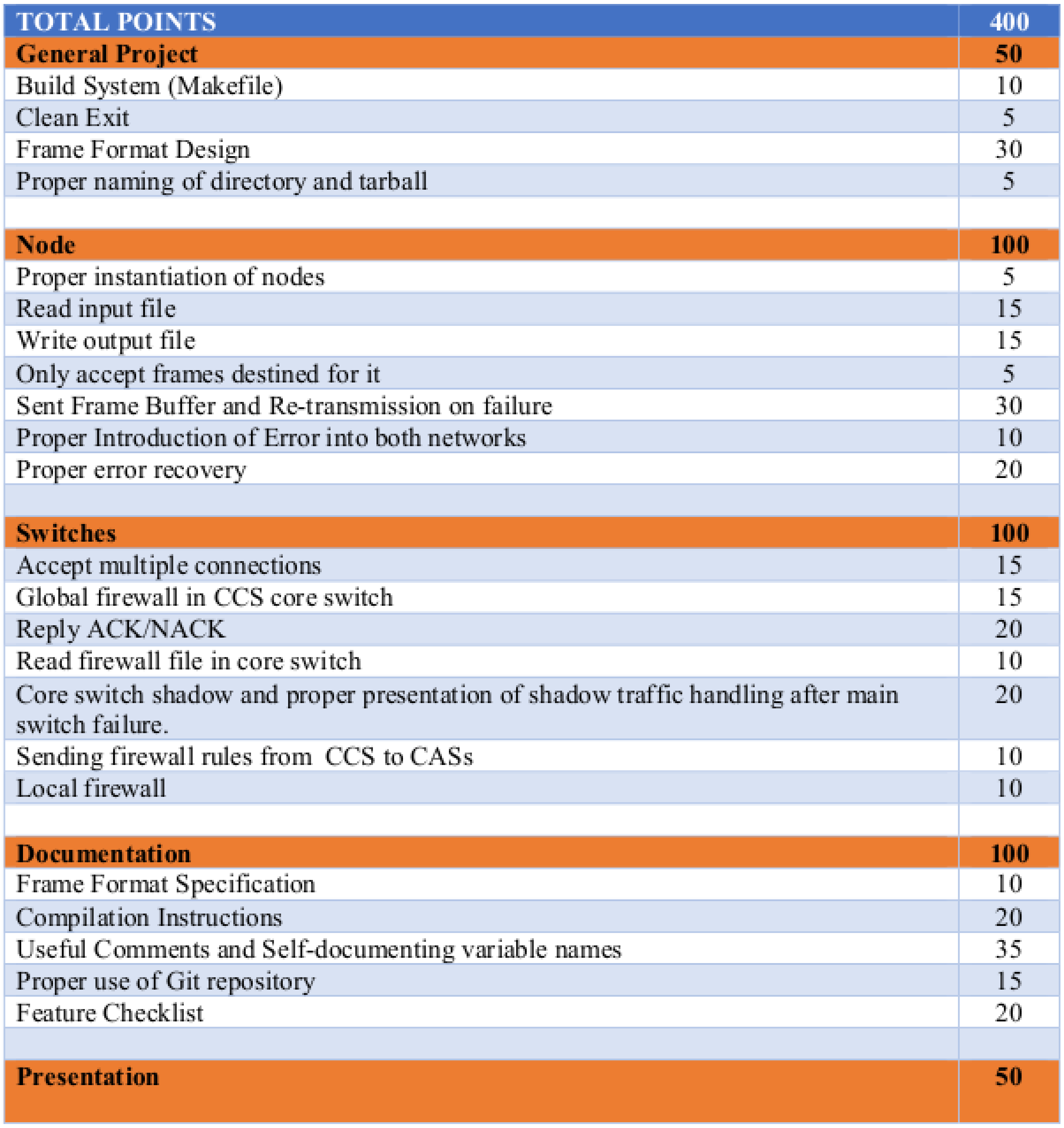
CSE 353 Project Star of Stars Network – A.S. (solo)

IMPORTANT:
As no way to generate nodes and node config txts was given, some node configs are are provided in the files, it is 5 CAS and 10 nodes (2 per CAS), as well as another two nodes (node 6_1 and 7_1) that were used for intial testing. The instantiation of the nodes is hardcoded in tests.py, where a thread per each is made. If nodes are instatiated with the proper parameters for the constructors, there should be no problem running different nodes. Refer to tests.py to see the details.

SUMMARY OF WHAT WAS DONE IS BELOW



TOTAL POINTS	**400**
General Project	**50**
Build System (Makefile)	10
Clean Exit	5
Frame Format Design	30
Proper naming of directory and tarball	5
Node	**100**
Proper instantiation of nodes	5
Read input file	15
Write output file	15
Only accept frames destined for it	5
Sent Frame Buffer and Re-transmission on failure	30
Proper Introduction of Error into both networks	10
Proper error recovery	20
Switches	**100**
Accept multiple connections	15
Global firewall in CCS core switch	15
Reply ACK/NACK	20
Read firewall file in core switch	10
Core switch shadow and proper presentation of shadow traffic handling after main switch failure.	20
Sending firewall rules from CCS to CASs	10
Local firewall	10
Documentation	**100**
Frame Format Specification	10
Compilation Instructions	20
Useful Comments and Self-documenting variable names	35
Proper use of Git repository	15
Feature Checklist	20
Presentation	**50**

Summary of implemented features according to table above

GENERAL PROJECT:

Build System (Makefile): The tests.py file can be executed if it is the same directory as the other files.

Clean Exit: Not yet implemented, only CCS and CAS shut down, not the nodes.

Frame Format Design: partly implemented, is used throughout the code and can be verified in frame.py and framedefs.py

Proper naming of directory and tarball: Will be named schwierz_CSE353_Project3.tar.gz

NODE

Proper instantiation of nodes: all nodes instantiate as separate threads in tests.py

Read input file: nodes read their input files

Write output file: if a node receives the frame, it will write to the output. This seems to vary from run to run. With the test nodes I am providing, they write mostly fine.

Only accept frames destined for it: yes, frames that don't have the nodes id as destination are discarded.

Sent Frame Buffer and Re-transmission on failure: retransmitts a few times until timeout.

Proper Introduction of Error into both networks: Not sure, as I am not sure what this means

Proper error recovery: likely not implemented

SWITCHES

Accept multiple connections: Yes, all CAS and the CCS accept multiple connections.

Global firewall in CCS core switch: Configuration is read and intilialized in CCS the readConfig method of switchCS.py, it is stored as a variable of the instance, but it is not yet used.

Reply ACK/NACK: Not yet implemented

Read firewall file in core switch: Firewall is read and stored in core switch

Core switch shadow and proper presentation of shadow traffic handling after main switch failure: swicht failure handling not implemented

Sending firewall rules from CCS to CASs: not sent

Local firewall: not done

DOCUMENTATION

Frame Format Specification: Can be found in the class frame.py, the offsets for the frame fields are constants in framedefs.py

Compilation Instructions: Paste all files into the same folder, open a terminal and run python3 tests.py (**NOTE:** for multiple runs, as the termination isn’t proper yet, the port number for the CCS switch can be changed in ccsdefs.py, which enables running again on a different port)

Useful Comments and Self-documenting variable names: will work on that until submission
Proper use of Git repository: GIT link is https://github.com/AlmondNMT/353Proj3.git. The repo is currently private, but will be published upon request to prevent plagiarism. I worked on the project alone.

Feature Checklist: will be included here

Feature	Status/Description
Project Compiles and Builds without warnings or errors	**Complete**
CAS, CCS Switches has a frame queue, and reads/writes appropriately	**Complete**, frame queues are called framebuffers and a property of the objects. They are lists containing frame objects. The socket is used as the byte buffer and exists as well. It is accessed regularly using s.recv() Frames are sent in binary across the wire
CAS, CCS Switches allows multiple connections	**complete**
CAS, CCS Switches flood frames when it doesn't know the destination	**Complete**, special frames with ack values 5 and 6 are used to prompt either a node to tell CAS where it is (value 6), nodes confirmation to CAS (value 6 also) or the same between CAS and CSS (value 5). CAS are implemented to automatically register with the CCS, because the port of CCS is unique and globally known.
CAS, CCS Switches learn destinations, and doesn't forward packets to any port except the one required	**Complete**, based on destination information.
CAS connects to CCS	**complete**
CAS receives local firewall rules	**Not done**
CCS switch opens the firewall file and gets the rules	**Complete**, is opened, read and stored in instance of CCS
CCS passes global traffic	**compelete**
CCS Shadow switches run and test properly	**Not done**
Node class	**Complete**, except for proper shutdown
Nodes instantiate, and open connection to the switch	**complete**

Nodes open their input files, and send data to switch.	**complete**
Nodes open their output files, and save data that they received	**Partially complete** – depending on the state of the network, some frames are lost sometimes. Not sure why. Behavior is inconsistent. Frames that are lost in one run may be properly written on the next one.
Node will sometimes drop acknowledgment	May occur, but not on purpose
Node will sometimes create erroneous frame	Not done
Node will sometimes reject traffic	May occur, but not intentionally