CSE/IT 353 **PROJECT #3** **Due date:** Friday, Dec 2nd, 11:59pm

STAR of STARS (SoSs) with BRIDGE and FIREWALL

Project Goal

The aim of this project is to implement a Star of Stars (SoSs) Network, using TCP/IP sockets, as a continuation of Project #2. The designed SoSs network will have a Central Star (CS) that connects a number of Arms-end Stars (ASs), see Fig. 1 (e.g., Speare, Cramer, Weir LAN stars). Each AS will function as in Project #2, with the following additional tasks. Each Core switch of an AS (CAS) will bridge global traffic frames in its own star to the Core of the CS (CCS), i.e. global switch (ITC in Fig 1). Then, the CCS inspects and segregates traffic forwarding, based on a preloaded ***firewall*** table, set up by the Network's admin (you!) at the protocol initialization phase, which decides "**who can talk to whom**". As a result, it might forward or block (for security reasons) the inspected frame. In case of accepting the frame to forward, the CCS will forward each global frame to its target destination remote star CAS (Speare, Cramer, and Weir in Fig. 1). The global frame forwarding is based on a preloaded Global Firewall Table, GFT (at the initialization phase) that contains all nodes in the SoSs Network and their associated CASs.
Your SoSs Network protocol should include the CCS core ***firewalling*** and ***bridging*** functionalities/mechanisms.
Initially, to maintain **local ASs security**, the CCS will also forward firewall tables to all CASs in the SoSs Network, initialized by the network admin. Hence, local traffic segregation for each AS is also maintained. In addition, to overcoming the inherited vulnerability of attacking the CCS, as well as the CASs, you should address in your protocol a remedy for such **robustness problem** for CAS's via the ***shadow backup*** solution, as you already did for the CASs in Project #2. Moreover, following the steps of Project #2, you will instantiate the ***node*** (**ASs nodes**) and ***switch*** (core of **stars**) objects that will connect to the local stars via the **$\text{CAS}_{s/d}$**, which will then allow inter-node global/local frame communication.

Project Description

Motivation

In large networks, engineers will deal with bridging LANs of varying regimes, such as different ***acknowledgement*** schemes, transmission ***speeds***, ***packet formats***, and ***firewalling***. In your protocol design of SoSs Network, you should implement the aforementioned functionalities (challenges).
Additionally, in any star network, there is a key drawback which is a central bottleneck. One way to alleviate such a problem is the introduction of a ***shadow*** switch. It is supposed to be always "*in-sync*" (synchronized) with the active/original switch to be able to take over immediately upon its failure.
Local traffic of ASs is going to be handled using the local ASs switches; however the forwarding of the inter-ASs traffic will be handled by the CCS node. The CCS node is also responsible for ***firewalling*** and ***forwarding/bridging*** the firewall rules to other local switches to handle local firewalling functionalities.

Overview

You are to create an object-oriented program that will spawn objects and threads. You need to create multiple nodes and switches, at the ASs and higher-level global star switch which we named above as CCS of the SoSs Network (ITC in Fig 1). Each switch will use a listening socket to allow nodes to connect to it, and spawn worker threads as needed to allow inter-node communication, globally (ITC) and locally (Weir, Cramer, Speare). The CCS switch will act like a modified version of the CAS switch to work globally (all SoSs), instead of locally (individual AS).
The ASs of SoSs will connect to their associated network core switch (CAS) and will send local and global frames. Each AS node will open a file, and send data (or possibly buffer for future retransmission) to other nodes via their associated local CAS switch, which will forward the local traffic to another local node, and global traffic to the CCS

switch. The distinction of *local* versus *global* traffic is based on a preloaded (at initialization phase) forwarding table listing all local node addresses at the CAS switch. It is also possible that the CAS blocks traffic from and to specific source/destination address based on a preloaded firewall table (provided by the CCS switch, ITC in Fig 1, at the initialization phase).

Traffic Forwarding:

Local at AS's:

At each AS, when a source node (SN) transmits a frame, it will buffer it, then pause for some time-out period, waiting for ACK back from the frame destination. Meanwhile, the CAS will receive and check the transmitted frame's destination/source addresses. The CAS will look up the frame's source address (local or global) – if local and also **safe** to forward to the destination **local** node, then it will forward it. Yet, if it is rejected based on the CASs *firewall* table, then the CAS will send back a Negative ACK (NACK) to the awaiting source node There should be a max number of re-transmissions of the frame, before deleting it from buffer and reporting "error" to ***SoSs's*** Network admin.

Global Traffic via CCS:

In case of the CAS not finding the transmitted frame destination address in its forwarding table, then it will forward it to the CCS. The CCS upon receiving the global frame carries out a similar firewalling process as at the local CAS, but at the inter-star level (among Cramer, Weir, Speare in Fig 1). If the frame is denied the forwarding, based on the CCS firewall table, the CCS will send back NACK to the source CAS, which in turn passes it to the awaiting source node (the rest is the same as in local traffic). But if the frame is granted forwarding, the CCS will pass it to its target CAS, where its destination address resides, based on its pre-stored GFT. When the destination CAS receives the forwarded frame from the CCS, it forwards it to its target destination node. In case the destination node sends a good ACK back to its local destination CAS, it will be forwarded to the awaiting CCS, which forwards it back to the awaiting source CAS. Upon receiving the ACK from the CCS, the source CAS will send the ACK to the source node (the rest is the same as in local traffic). See flowchart as appendix.

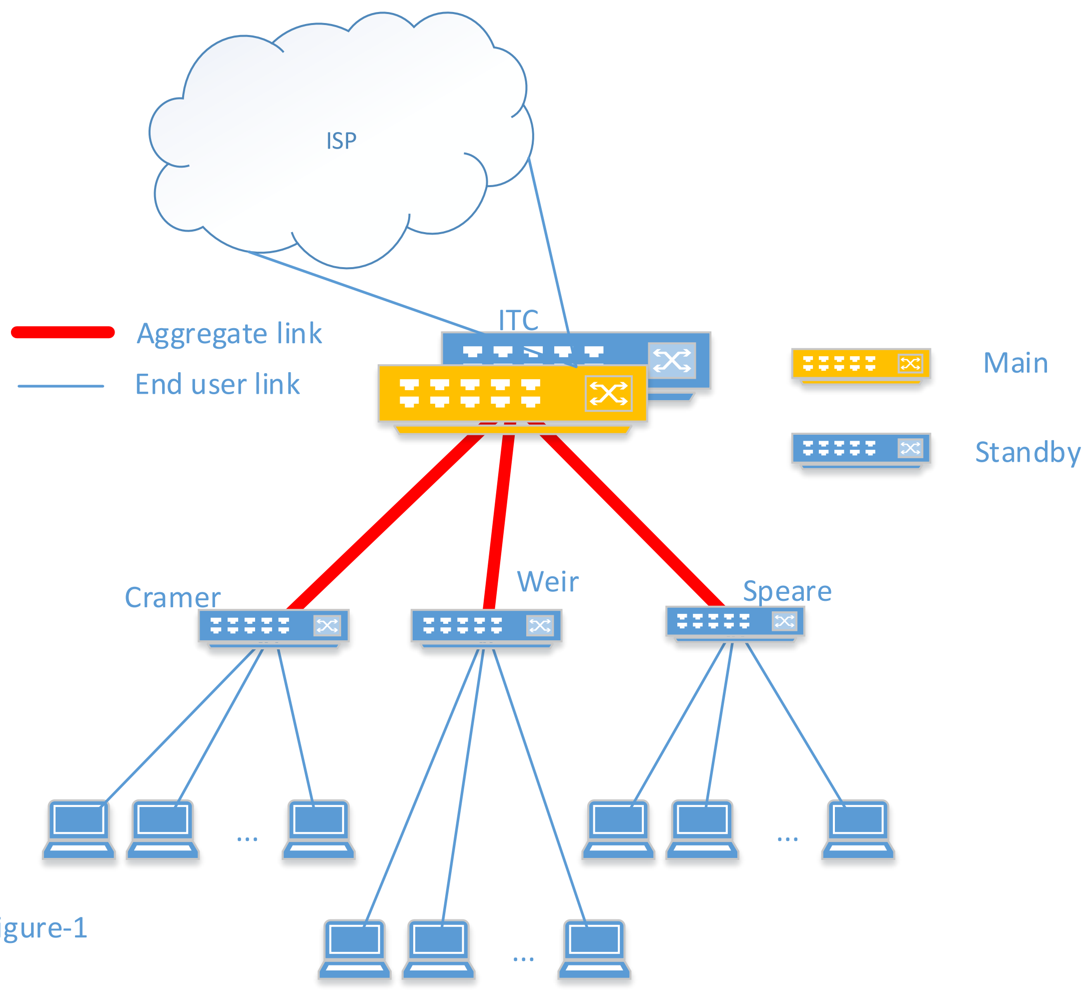


Figure-1

Main Class
Your main class should instantiate some nodes and switches and a single core switch, the number of which should be read from the command line arguments. It should also wait until all nodes are done sending data, before shutting down all the individual nodes, followed by the switch and hub, and exiting cleanly.

CAS Switch Class
The switch in this project will operate almost identically to the switch in the last project. It should allow multiple connections, and use frame flooding and source address matching to determine where to send frames. The switches in this project also need to connect to the core switch to be able to forward global traffic to it.
Additionally, they should be capable of firewalling and bridging functionalities. All switches take firewall rules from the CCS switch at start-up.

CCS Switch Class
This switch is mostly similar to the CAS local switch except it is more powerful and faster to be able to handle the aggregated traffic. Initially, CCS is going to flood frames to the underlying CAS instead of end users, and then it can just forward the traffic based on the network number, with no need to process the end user address number.
It should also take care of executing and forwarding firewalling rules. The CCS switch should read the firewall rules from a file, which is going to be given as in the file format section. The CCS should operate the internetwork traffic firewalls. CCS should also transmit the local firewalling rules to CASs. You may use specific data format or frame field to forward the firewall rules to CASs.

CCS Shadow Switch
It's the same as the CCS switch, including the updated forwarding table. It will start traffic forwarding at the moment the main CCS switch fails, which is highly probable in real networks due to power failure or any other hardware or link failure.

Node Class
The nodes are similar to the nodes from Project #2. I encourage you to use as much of your Project #2 code as possible. The nodes should connect to their associated network switch. The nodes are to be numbered (x, y), in which x is the network number and y is the local node number. You should design your code so that graders can instantiate between 2 and 16 nodes easily.
Every node should open and read an input file, which contains data that it should send across the network to other nodes (See Files section below). Every node should also create an output file which contains all the data that was sent to it by the other nodes. The nodes should send data to the other nodes via local switch, and the local switch should forward the packet to the local destination or forward it to the core for global traffic. The node should keep a copy of the frame it sent in a buffer, in case the frame is corrupted along the way and the sender has to re-send it.
When the data is successfully received, the node that received it should send back an acknowledgment to the sender via control bits as specified in the frame format (see Frame Format section below). When the acknowledgment is successfully received, the frame can be removed from the sender's buffer.
Each of the nodes should have a **5% chance of creating an erroneous frame** on the networks. In addition, the nodes should also have a **5% chance of failing to acknowledge** a frame on networks.

Files
A script will be provided that will generate data files for the nodes to send. The files will be named nodex_y.txt, where x is network number and y is the node number. The nodes will open their files, and send the data to the other nodes. The following is an example input file for node2_1:

```
1_2: ABCDEFG
3_4: 1234567
2_6: This data will be sent to node 2_6.
```

In this example, node2_1 will send `ABCDEFG` to node1_2 which is global traffic and is going to go through the core switch. You may assume that no node will want to send data to itself. The nodes are responsible for creating a file, named nodex_youtput.txt, which will contain all the data that was sent to it, and who sent that data. Using the previous example file, Node1_2 will create `node1_2output.txt` which will have the line '2_1: ABCDEFG` which means it received the data line from node2_1.

Firewalling file:
The following file is going to be read by CCS switch and be executed by itself for global traffic and also forwarded to the CAS local switch to let them do it locally.

3_5: **local**
2_#: **local**

X_# Local means that the CAS just accepts local traffic, but global traffic is supposed to be blocked by CCS from forwarding to this CAS.
X_Y Local means that the specific node just accepts local traffic, but global traffic is supposed to be blocked by its CAS from forwarding to this node.

Frame Format
Data Frame:
This network will use the same data frame format as the previous project, with the addition of the CRC field.

[DST][SRC][CRC] [SIZE/ ACK][ACK type][data]
DST: Destination of the frame, 1 byte, 1-255
SRC: Source of the frame, 1 byte, 1-255
CRC: Cyclic redundancy check, 1 byte
    The CRC field should contain the sum of the byte values of the frame including header and Data, truncated to one byte. This provides a checksum that a destination node can check to ensure that the data arrived intact. When you calculate CRC, you should make CRC as 0x00 (or NULL) at the beginning. After you get the CRC result, fill the result into CRC field.
SIZE/ACK:
    In a data frame, this field has the size of the data in bytes, from 1-255
    When size is 0, the data field is omitted, and this is treated as an ACK.
ACK type:
    It will be filled, only if the frame is an ACK, 1 byte.
data: The actual Data (1-255 bytes)
End of frame

ACK type: in this field we have
- 00 no response in time out (**resend again**).
- 01 CRC error (**resend again**).
- 10 firewalled (**no need to resend**).
- 11 positive ACK.

Project Requirements

Language – You are required to implement the project in one of the following programming languages: Java, C#, C++ or Python.

Environment – Your program must compile and run on the CS machines (login.cs.nmt.edu). All projects will be graded on the CS machines. If your project does not compile or run on this system, then your grade will suffer.

Build Automation – You are required to employ some sort of build automation for this project. Acceptable formats are GNU make for C++, C#, Java or Python (Standard MakeFile).

Documentation – You need to create a README file that includes the following:

a) Names of all group members.
b) Git repository link.
c) How to compile and run your program.
d) The names of all files in the project and a brief description of their purpose.
e) Provide a checklist that includes which features are implemented, and which features are missing. The minimum required checklist for this project is below.
f) A list of all known bugs. Documenting bugs will reduce the corresponding point deduction.

Code Style – Source code should be well-organized, follow accepted conventions for that language, and be well-documented with meaningful comments and variable names.

Frame Format – The frame should not be sent as a serialized object, or as a simple string across the wire! The frame should move in a binary format across the wire, using the frame specifications laid out above. Additionally, any deviations from the specific frame format requires documentation!

Assumptions

The following is a list of assumptions you are allowed to make in regards to your program:

1. You are not required to verify the format of the input files, since they are guaranteed to strictly follow the format defined above.
2. You are not going to have more than 16 nodes connected to any CAS at any given time.
3. A single frame of data will not contain more than 255 bytes.
4. No node will attempt to send data to itself.
5. You are not required to provide a GUI of any kind. Output to the console is both acceptable and encouraged.

Feature Checklist

Include a specification of the frame format in your README. This should include the order of the fields, the byte sizes and acceptable ranges for the fields, and a short description of the function of the field.

Include the following checklist in your README. Each item should be marked as complete, missing, or partial. In the case of a partial status, make sure to add a description as to what works and what does not. The following is an example; it is expected that you will change the "Status/Description" column to indicate progress in your project.

Feature	Status/Description
Project Compiles and Builds without warnings or errors	complete
Switch class	complete
CAS, CCS Switches has a frame queue, and reads/writes appropriately	complete
CAS, CCS Switches allows multiple connections	complete
CAS, CCS Switches flood frames when it doesn't know the destination	complete
CAS, CCS Switches learn destinations, and doesn't forward packets to any port except the one required	Incomplete. Switch acts like a hub.
CAS connects to CCS	complete
CAS receives local firewall rules	complete
~~**CAS sends AC=00 back**~~	~~complete~~
CAS forwards traffic and ACKs properly	complete
CCS switch opens the firewall file and gets the rules	complete
CCS passes global traffic	complete
CCS does the global firewalls	complete
~~**CCS send AC=00 back.**~~	~~complete~~
CCS Shadow switches run and test properly	complete
Node class	Partially Complete – see below
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	Partial – They also save flooded frames
Node will sometimes drop acknowledgment	complete
Node will sometimes create erroneous frame	complete
Node will sometimes reject traffic	complete

Project Presentations
Like the second project, you will be required to present your project to both of the class TAs, at a time period outside of class (by appointment slot). The presentation is to be short (between 5 and 10 minutes), and it should showcase your features, your challenges, how you overcame them, and your bugs, as well as a demonstration. All group members should participate in the presentation. The presentations will be held after the final project submission date.

Project Submission
Place all of your source code, your build files (makefile or build.xml), and your README, into a directory named "<Group#><Lastname1>_<Lastname2>_<Lastname3>_CSE353_Project3" and tarball the directory. The README file should also have Git log statistic information per member. The tar archive should also be named using the same convention: "<Group#><Lastname1>_<Lastname2>_<Lastname3>_CSE353_Project3.tar.gz". Zip files are also acceptable. Submit your archive file to Canvas before the deadline. Please see the class syllabus regarding late assignment policy.

Academic Honesty
ALL WORK MUST BE YOUR OWN! YOU MUST CITE ANY CONTRIBUTIONS THAT YOU RECEIVE FROM CLASSMATES OR ANY OTHER EXTERNAL SOURCES. This doesn't include contributions from your group members.

Contact Information

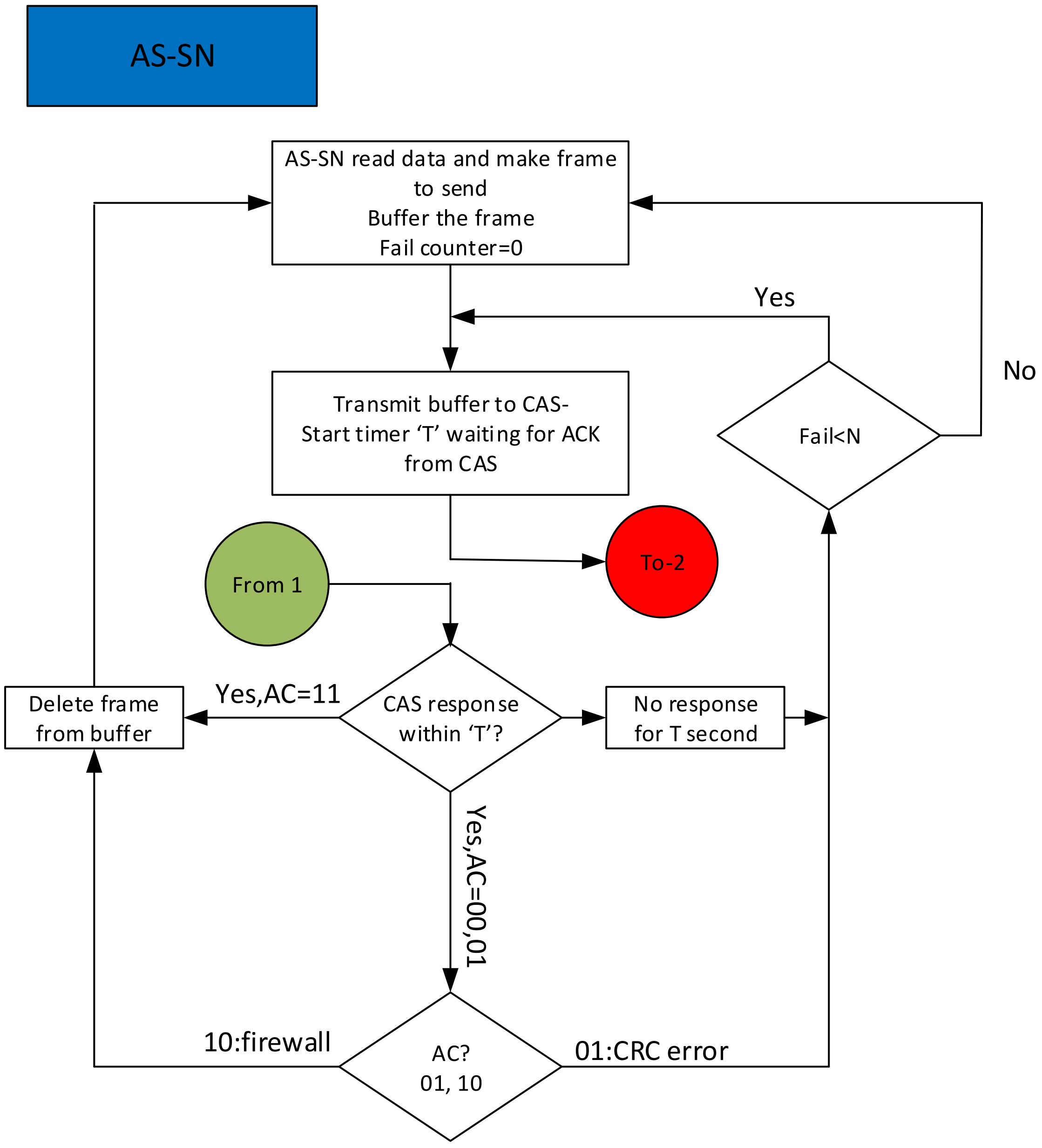
If you have questions, please send a message through Canvas to the class TAs, Mohammad Hossen & Tanjim. If this fails for some reason, please send via email to mohammad.hossen@student.nmt.edu, tanjim.fatima@student.nmt.edu, or stop by TA office hours.

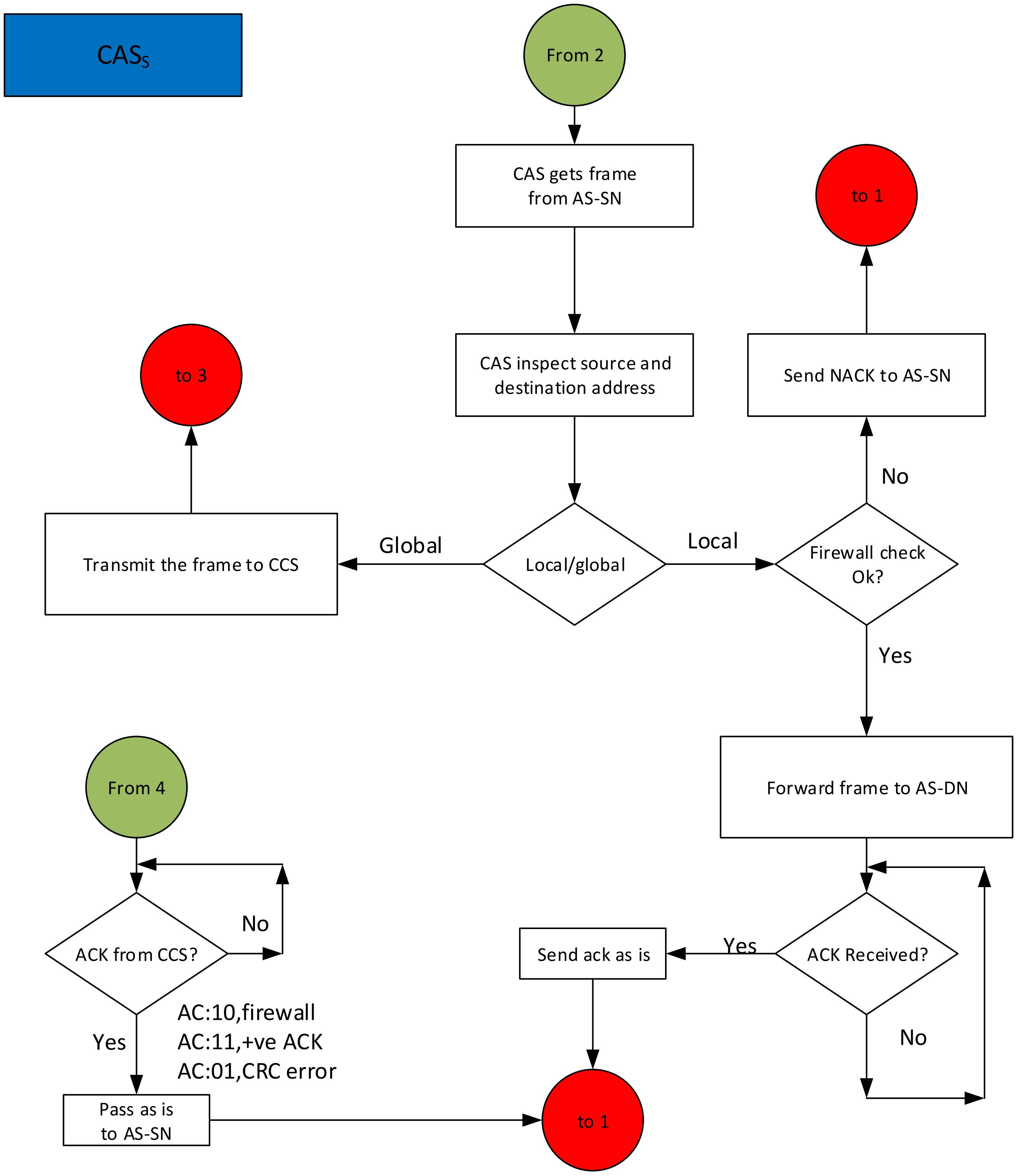
Grading

The project will be graded according to the following rubric:

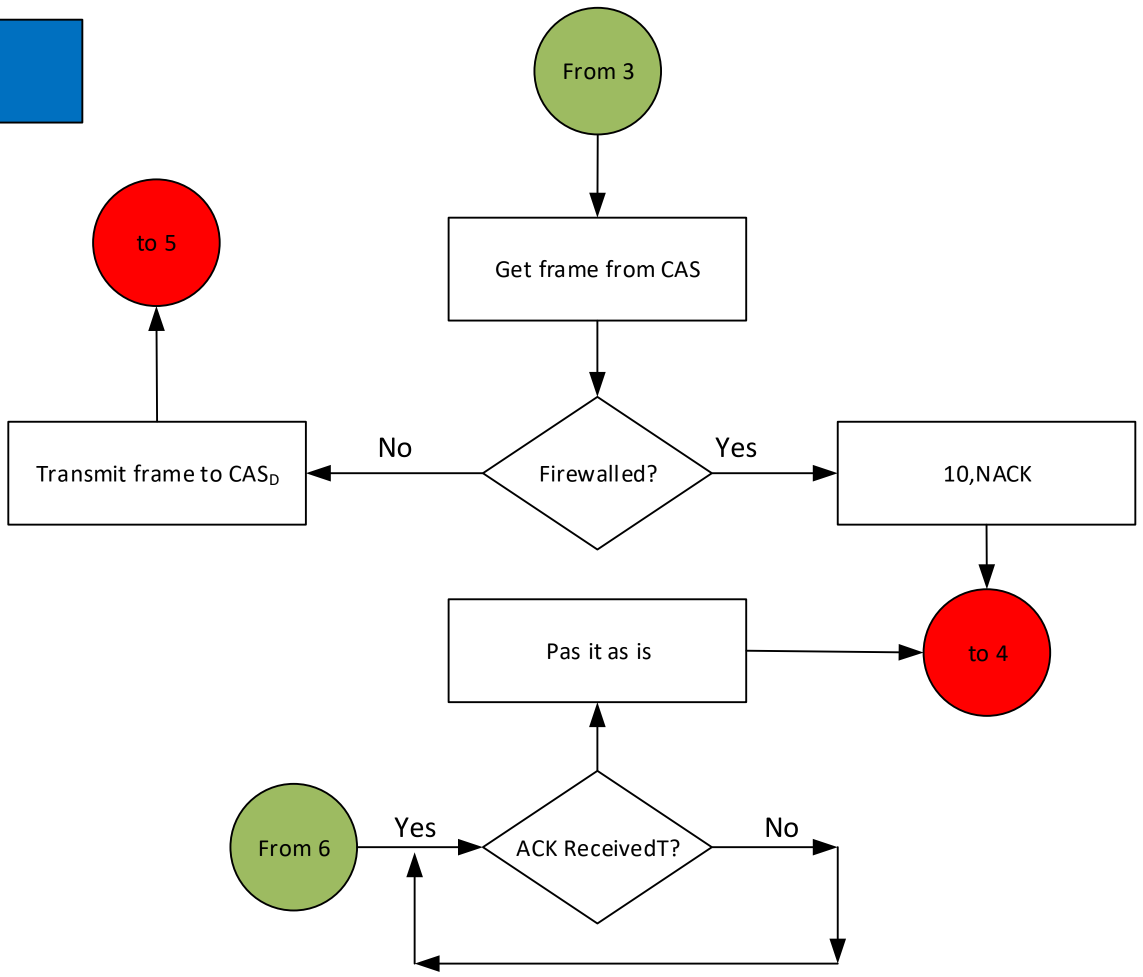
TOTAL POINTS	**400**
General Project	**50**
Build System (Makefile)	10
Clean Exit	5
Frame Format Design	30
Proper naming of directory and tarball	5
Node	**100**
Proper instantiation of nodes	5
Read input file	15
Write output file	15
Only accept frames destined for it	5
Sent Frame Buffer and Re-transmission 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**

APPENDIX: FLOWCHART



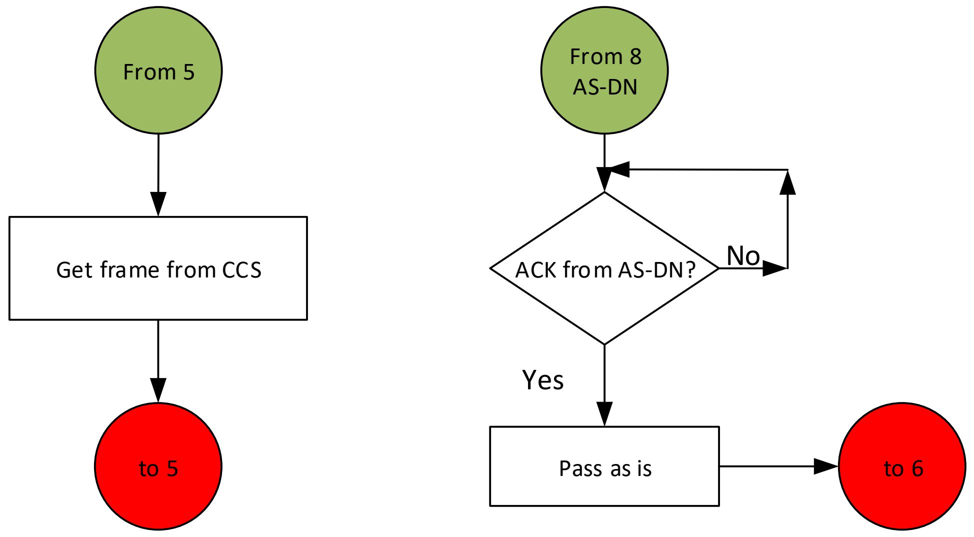

CAS$_S$
From 2
CAS gets frame from AS-SN
CAS inspect source and destination address
to 1
Send NACK to AS-SN
No
to 3
Transmit the frame to CCS
Global
Local/global
Local
Firewall check Ok?
Yes
Forward frame to AS-DN
From 4
ACK from CCS?
No
Yes
AC:10,firewall
AC:11,+ve ACK
AC:01,CRC error
Pass as is to AS-SN
Send ack as is
Yes
ACK Received?
No
to 1

CCS



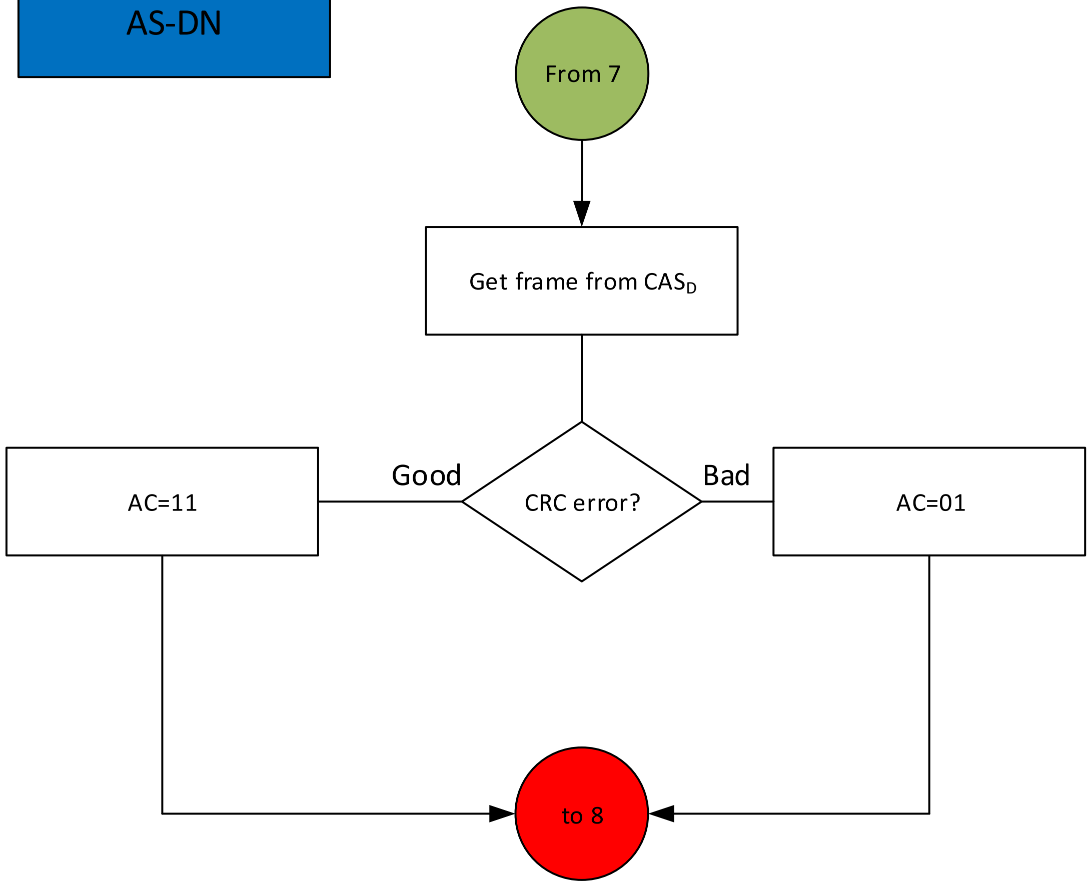
From 3
Get frame from CAS
Firewalled?
No
Yes
Transmit frame to CAS_D
to 5
10,NACK
to 4
Pas it as is
From 6
Yes
ACK ReceivedT?
No

CAS_D



From 5
Get frame from CCS
to 5
From 8 AS-DN
ACK from AS-DN?
No
Yes
Pass as is
to 6

AS-DN


From 7
Get frame from CAS_D
CRC error?
Good
Bad
AC=11
AC=01
to 8