Lab 4

Byzantine Agreement

1 general per node, goal: Byzantine Agreement

(honest, vote$_1$)

Each general decides on a profile ("honest" or "Byzantine"). Honest generals have an initial vote ("attack" or "retreat"). Byzantine generals will try to break the agreement.

(honest, vote$_2$)

(honest, vote$_3$)

All generals run the Byzantine Agreement alg. Honest generals should agree on a result (attack or retreat)

Byzantine

How it should work

- N nodes, each general controls one node

- For each general (node)
  - Decide on a profile "honest" or “Byzantine”
  - For honest generals decide on a vote “attack” or “retreat” and
  - start the Byzantine agreement algorithm (see lecture "Fault Tolerance I") on all generals, through a webpage interface (we provide the interface – see next slide)

- When the algorithm ends, the **result and result vector** (see slide "Byzantine Agreement Problem (8)" on lecture "Fault Tolerance I") should appear on each webpage (upon refresh)
  - Honest generals should agree on the same votes among them
  - We don't require any agreement for the Byzantine generals

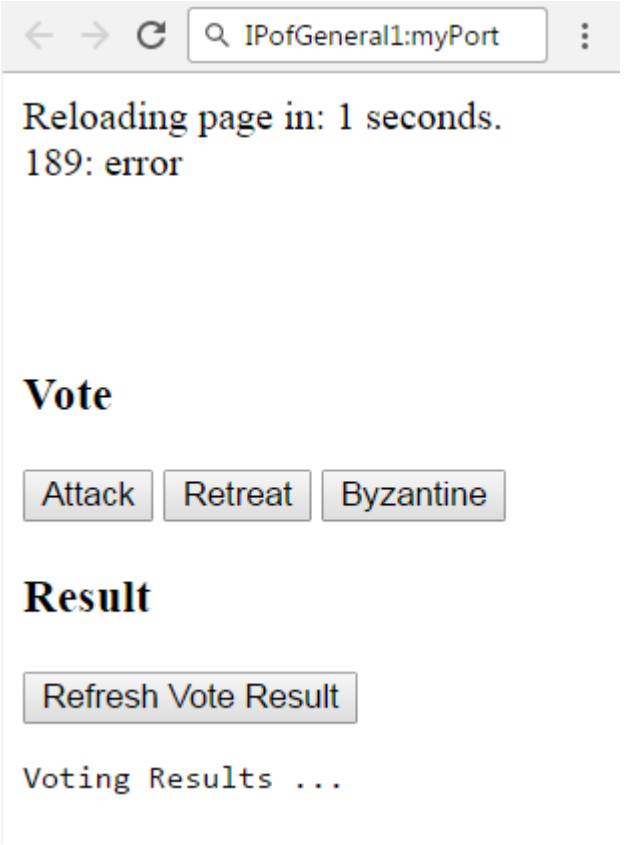
The interface

- Each node has a webpage interface (you can use the blackboard page if you like) of three buttons:
  - “Attack”, “Retreat”, “Byzantine”

- If a general is honest, start the Byzantine agreement algorithm on that general (node) by clicking the “Attack” or “Retreat” button, depending on the initial vote that you have assigned to that general

- If a general is Byzantine, start the Byzantine agreement algorithm on that general (node) by clicking the “Byzantine” button

- Code is explained/provided in the Appendix



What to do

- Use the webpage interface to initialize the Byzantine agreement algorithm for each general (node). Recall that initially generals are unaware of each others votes. Each of the “Attack”, “Retreat”, and “Byzantine” buttons should call a callback function (as in post request, etc.) that initiates the Byzantine agreement algorithm (only) on that general (node).
- When the callback is called with argument “Attack” or “Retreat”, your code should handle the behavior of an honest general that votes attack or retreat, respectively
- When the callback is called with argument “Byzantine” your code should handle the behavior of a Byzantine general (we provide the code for that – see Appendix)
- All nodes are aware of the total number of nodes. Only the Byzantine nodes know which nodes are Byzantine (i.e., their IPs).

What to do (cont.)

Byzantine agreement algorithm runs in two steps (see lecture “Fault Tolerance I”):

- (step 1) When voting starts:
  - honest nodes start by sending out their votes
  - Byzantine nodes wait until they collect all the honest votes and send out different votes to the honest nodes in order to break agreement (if possible).
- (step 2) When a node has received all votes
  - if honest, it sends to other nodes a vector of all votes received
  - if Byzantine, it sends to other nodes a vector of Byzantine votes
    - We provide the Byzantine behavior (see Appendix)
- Voting and outcome:
  - when a node has received all the messages from step 2, it computes the (majority vote) result vector and the result
  - Then, it adds the **result vector and the result** to the webpage (to be seen upon refresh)

What to do (cont.)

- A node should have different behavior when honest or Byzantine
- Byzantine code (`byzantine_behavior.py`) can be found in Lab 4's page in pingpong and its explanation in the Appendix. The same code works for all the tasks of this lab, but feel free to edit/improve the code (and the interface)
- Recall the "3k+1 rule":
  - when having a total of N = 3k+1 nodes, agreement can be reached only if at most k out of them are Byzantine
- For this lab, you should implement the following scenarios (tasks)

Task 1A

- Select 4 nodes for this subtask. Using the interface set:
  - 3 honest nodes and 1 Byzantine (N=4, k=1)
  - You can change the number of nodes by modifying lab1.py, line 134, or calling lab1.py with different arguments.
- Demonstrate that agreement is reached. That is, no matter what the honest nodes vote for, agreement can always be reached
  - the result vectors of the honest generals should match on the entries corresponding to the honest generals
  - Hence, the honest generals agree on the same result.
- Note that the Byzantine node must respect the agreement protocol, but it can change the votes to be sent to the honest nodes (e.g. it cannot sent garbage – in this implementation, not in general).

Task 1B

- Select 3 nodes for this subtask. Using the interface set:
  - 2 honest nodes and 1 Byzantine (N=3, k=1)

- Set different votes for the two honest nodes. The Byzantine node must be able to convince one node to attack and another one to retreat. Cf. example on the ”Fault Tolerance I” lecture slides.

- As in task 1A, the Byzantine node can only change the votes to be sent to other nodes, but always respects the agreement protocol.

What to explain in the video (or report)

In addition to your demo for tasks 1A and 1B, briefly discuss the following:

1. Give a high level description of the 3-4 main functions that you use in your implementation, while showing your code (at most 1-2 minutes)
2. Show/explain the part of your code that implements the agreement part of the protocol (about 1 minute)
3. Show/explain how you implement the 2 phases of the Byzantine agreement protocol (in the asynchronous setting of Seattle)
   (about 1 minute)

Please try to be brief, but precise! :)

Extra reading

- Here is a link to the original paper by Lamport et al.: http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf

- There are many more resources about Byzantine agreement on the web (for various versions of the problem)

- This lab is tied to the Byzantine agreement problem as it is described on the lecture slides

APPENDIX

Simplistic HTML Interface

- Three buttons
- Show results as simple text
- Page reloads periodically

Functions	API	Param.	Returns
Vote: Attack Vote: Retreat Vote: Byzantine	POST /vote/attack POST /vote/retreat POST /vote/byzantine	None	Status
Get result	Get /vote/result	None	Vote results after Byzantine agreement. Show Individual nodes results and final result.
Show homepage	GET /	None	Homepage

HTML Template

- Update your board_frontpage_header_template.tpl file with the one found in the ping-pong page of Lab 4

Byzantine behavior

- We provide you the simple functions that implement the logic of the byzantine nodes, on a .py file on pingpong.
  - You just put them in your code and call them as described next.
- On round 1, byzantine node calls:
  - compute_byzantine_vote_round1(#loyal_nodes, #total_nodes, tie-braker)
    - tie-braker is a boolean variable (True or False) indicating Attack or Retreat respectively.
  - Result: A list with the votes to send to the loyal nodes e.g. [True,False,True,….]

Byzantine behavior (cont.)

- On round 2, byzantine node calls:
  - compute_byzantine_vote_round2(#loyal_nodes, #total_nodes, tie-braker)
  - Result: A list, where every element is the vector to send to each loyal node.
  - e.g. [[True,False,…] , [False,True,…] , …]
  - Again, True stands for Attack and False stands for Retreat.