Virtual Beamer

Java application to synchronously share a set of slides over LAN

Lorenzo Poletti and Mariusz Wiśniewski

July 8, 2022

Overview

1. Requirements
2. Assumptions
3. Development Tools & Frameworks
4. Implementation
5. Summary

Requirements

Requirements

Virtual Beamer is a *Java* application to share a set of slides and display them in a synchronous way under control of a single node (the leader).

- In order to share a presentation, the user needs to create a new session.
- Other nodes join one of the available sessions.
- There can be more sessions led by different users.
- When the session creator decides to share the slides, the presentation may start.
- Users (nodes) may join while the presentation is ongoing. In such case, they choose the node from which to download the presentation in a way to share the load.
- The leader (initially the session creator/owner) decides on the slide to be displayed.
- During a session, the leader may pass its role to another user, which subsequently becomes the new leader.

Assumptions

Assumptions

- LAN scenario (i.e., link-layer broadcast is available).
- The nodes (including the leader) may crash. In such case, the leader becomes the session creator, or the leader is elected if the session creator also crashed.

Privilege a *plug&play* solution that leverages the LAN scenario to avoid the need of entering network addresses and similar information.

Development Tools & Frameworks

Development Tools & Frameworks

Development

- *Java SDK 18*
- *JavaFX 18*
- *Apache Maven 4.0*

Testing

- Running the application on multiple computers connected to the same LAN.
- Multiple instances of the application running on *Docker* containers with unique IP addresses and *X11 Forwarding* turned on.
- Multiple instances of the application running on the same computer with loopback mode turned on.

Implementation

Discovery of available sessions

- Periodically broadcast *HELLO* message using the global port.
- Wait for the leaders to respond with *SESSION_DETAILS*.

Create session

- Insert username.
- Provide the name of the session.
- Create a new group by setting its multicast port as the greatest received value + 1.
  - There has to be at least *PORT_TIMELINESS* number of *HELLO* messages sent to assume that all the ports have been collected.
  - Similarly, if the port has not been collected after the last *PORT_TIMELINESS* number of *HELLO* messages sent, it is assumed that it is no longer occupied.
- Stop sending *HELLO* message.
- Broadcast *SESSION_DETAILS* using the global port.
- Start periodically multicasting *IM_ALIVE* inside the group.

Share slides

- The presenter creates byte packets from the slides (images) and multicasts **all** of them inside the group as soon as he loads them into the session.
  - The slides may exceed the *UDP* maximum payload. Therefore, they are split up into appropriately sized chunks before being transferred over the network.
  - Flags, slide number, and number of packets are included in the *UDP* packet to ensure that the image slices are reassembled into the correct sequence.
- The presenter multicasts *PREVIOUS_SLIDE* or *NEXT_SLIDE* inside the group to navigate between slides.
- Received slides are cached and displayed on the viewer's side.
  - The viewer caches all the received slides, so that when it becomes the leader, it can show them again.

Identify packet miss

- Every participant queue the incoming coming and every 2 seconds they are processed.
- If in the batch there are missing packets, these are requested through TCP to the leader.
- After the missing messages are received, they are sorted through ID and handled.
- Before empty the queue, the packets are added to the processed queue, which keeps track of the handled messages.

Join session

- Unicast *COLLECT_USERS_DATA* directly to the leader of the session to be joined and wait for *USER_DATA* messages unicasted by the leader.
- The viewer sets its ID as the greatest received value + 1.
- The viewer unicasts *JOIN_SESSION* with its username, ID, and IP address directly to the group leader.
- The leader multicasts *NEW_PARTICIPANT* together with the newcomer's data inside the group.
- If any of the *NEW_PARTICIPANT* message receivers has already received some slides, it begins the agreement process on who is going to provide the new viewer with the slides by multicasting *START_AGREEMENT_PROCESS* inside the group.

Agreement on the slide sender

- When a node receives the *START_AGREEMENT_PROCESS* message, it verifies if it knows about any participant with ID lower than the received one.
  - If so, then the node sends *STOP_AGREEMENT_PROCESS* message to the sender, which stops its timer. Then, starts its own agreement process by multicasting *START_AGREEMENT_PROCESS*.
  - Otherwise, does nothing.
- If *AGREEMENT_PROCESS_TIMEOUT* has been reached, the node with the lowest ID everybody agreed upon sends slides directly to the new viewer.
- In case, the viewer receives *PREVIOUS_SLIDE* or *NEXT_SLIDE*, and he still does not have the slides, he waits for *SLIDES_AVAILABILITY_TIMEOUT* and if he still does not have them, it is assumed that the agreement process failed, and he asks the leader for the slides directly by sending *REQUEST_SLIDES* via unicast.

Leader change

- The leader unicasts *PASS_LEADERSHIP* to the chosen user.
- The new leader multicasts *CHANGE_LEADER* message, which contains its data, to all the members of the group.

Crash detection

- When the leader join the session starts to send *IM_ALIVE* packets in broadcast.
- If a particpant doesn't receives *IM_ALIVE* packets for *CRASH_DETECTION_TIMEOUT*, it start the leader election.

Leader election
Bully algorithm

- When a node detects a crash of the leader (no *IM_ALIVE* message received for *CRASH_DETECTION_TIMEOUT*), it unicasts *ELECT* message with its *ID* to everybody in the presentation group and starts a timer.
- When a node receives the *ELECT* message, it verifies if its ID is lower than the sender's one.
  - If it is lower, then the node unicasts *STOP_ELECT* message to the sender, which stops its timer. Then, starts its own election by unicasting *ELECT* to everybody inside the group.
  - Otherwise, does nothing.
- If *LEADER_ELECTION_TIMEOUT* has been reached, the node multicasts *COORD* using the global port and starts periodically multicasting *IM_ALIVE*.
- Additionally, if the old leader rejoins the session, he automatically receives the list of all participants and all the slides from the active leader, and then he becomes the new leader through the *leader change* process.

Exit session

- If the leader exits the session, he unicasts *DELETE_SESSION* to every participant, gets redirected to the initial view, and starts periodically sending *HELLO* message.
  - Receivers follow suit.
- If a viewer leaves the session, he unicasts *LEAVE_SESSION* to the leader, who then multicasts *DELETE_PARTICIPANT* inside the group.

Summary

Summary

Get the source code of the application from:

`https://github.com/Nexer8/VirtualBeamer`