Virtual Beamer

Java application to synchronously share a set of slides over LAN

Lorenzo Poletti and Mariusz Wiśniewski

February 28, 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 on the presentation to share, 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 17*
- *JavaFX 17*
- *Apache Maven 3.8*

Testing

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

Implementation

Create session

- Insert username.
- Provide the name of the session.
- Multicast *COLLECT_PORTS* message using the global port.
- Create a new group by setting its multicast port as the greatest received value + 1.
- Multicast *SESSION_DETAILS* to using the global port.

Share slides

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

Discovery of available sessions

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

Join session

- The viewer multicasts *JOIN_SESSION* with its username and IP address inside the group.
- All the nodes that received *JOIN_SESSION* save the data of the new session participant and respond with *SEND_USER_DATA* message containing the username and ID.
- If any of the receivers has already received some slides, it begins the agreement process on who is going to provide the new viewer with the slides.

Identify packet miss

- This process is done in the group multicast.
- Each time the leader broadcast a message it saves the packet with its counter into a buffer.
- When a client discover that a packet is missing in its sequence starts a timer for sending a NACK message in group multicast. (The timer is related to the packet counter.)
- When a client receive the NACK message for a given packet counter stop the related timer if exists.
- If the leader receives the message, resend the packet associated to the counter.

Leader change

- When the leader presses give control while selecting an available participant it sends out a *CHOORD* message in multicast to all the members in the LAN informing the new leader name and IP of the session.
  - If who receives the packet is a member of the group then update the information in the current session. (Such as leader name displayed).
  - If who receives the packet is the new leader, than change role and update the graphic available components.
  - If is someone in the initial view, then update the list of available sessions.

Crash detection

- Every peer, leader excluded, start a timer when joining the session with random duration.
- When the timer stops, a *CRASH_CHECK* packet is sent in multicast to all the other peers and then waits a response for a fixed number of seconds.
  - If who received the packet is a viewer just restart the crash detection timer without sending any message.
  - If it's the leader reply with *IM_ALIVE* message.
  - If the leader doesn't reply within a certain amount of time is considered crashed and the bully leader election starts.

Leader election
Bully algorithm

- When a node detected a crash of the leader, it multicasts *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 sends *STOP_ELECT* message to the sender, which stops its timer. Then, starts its own election by multicasting *ELECT*.
  - Otherwise, does nothing.
- If ***LEADER_ELECTION_TIMEOUT*** has been reached, the node multicasts *COORD* using the global port.

Agreement on the slide sender

- When a node receives *JOIN_SESSION*, and it has already received some slides, it multicasts *START_AGREEMENT_PROCESS* message with the lowest *ID* of a node inside the presentation group he knows about to everybody in that group and starts a timer.
- When a node receives the *START_AGREEMENT_PROCESS* message, it verifies if it knows about any participants 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 sends slides directly to the new viewer.

Summary

Summary

Get the source code of the application from:

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