**Closing a Peer Connection**

When you call pc.close() on a WebRTC peer connection, it does not trigger a close event on the other peer's RTCPeerConnection object.

This means they must manage this situation through alternative means, such as monitoring the state of their own peer connection or handling the closure of individual data channels.

#### What about the onconnectionstatechange event?

While the connectionstatechange event is intended to indicate when a peer connection is closed by the other peer calling pc.close(), its behavior is not logical (to me, anyway). As you just observed in the previous lecture, instead of transitioning to a "closed" state, it moves to "disconnected" or "failed." This inconsistency necessitates the implementation of additional mechanisms to reliably detect when a connection has been closed by the other peer.

There are several approaches to achieve this.

Since we are utilizing data channels, we could listen for the close event on the RTCDataChannel, which provides a clear indication that the other peer has closed their data channel. However, since we are already using our signaling server to notify the other peer when a participant leaves, we can leverage this listener to also facilitate the closure of the other peer’s connection. The use of a signaling server to communicate the closure of a peer connection is a common practice amongst us developers in WebRTC applications.

#### Why the Difference?

If you will recall, dataChannel.close() triggers the onclose event on both peers' data channels. So then, the question arises, why the difference?

While I don't know the exact reasoning behind the design choices made by the creators of WebRTC, I can suggest a few possible explanations.

Firstly, data channels are designed to be independent communication channels that can be opened and closed without affecting the entire peer connection. This allows for more granular control over communication. In contrast, closing a peer connection is a more significant action that affects all associated streams and channels, and thus it does not provide the same level of notification to the other peer.

Secondly, different applications may have different requirements for how they handle connections and data channels. By not enforcing a specific behavior for peer connection closures, WebRTC provides developers with the flexibility to implement their own logic based on their needs.