INTER-SERVER GAME STATE SYNCHRONIZATION USING NAMED DATA NETWORKING

Philipp Moll, S. Theuermann,
N. Rauscher, H. Hellwagner

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

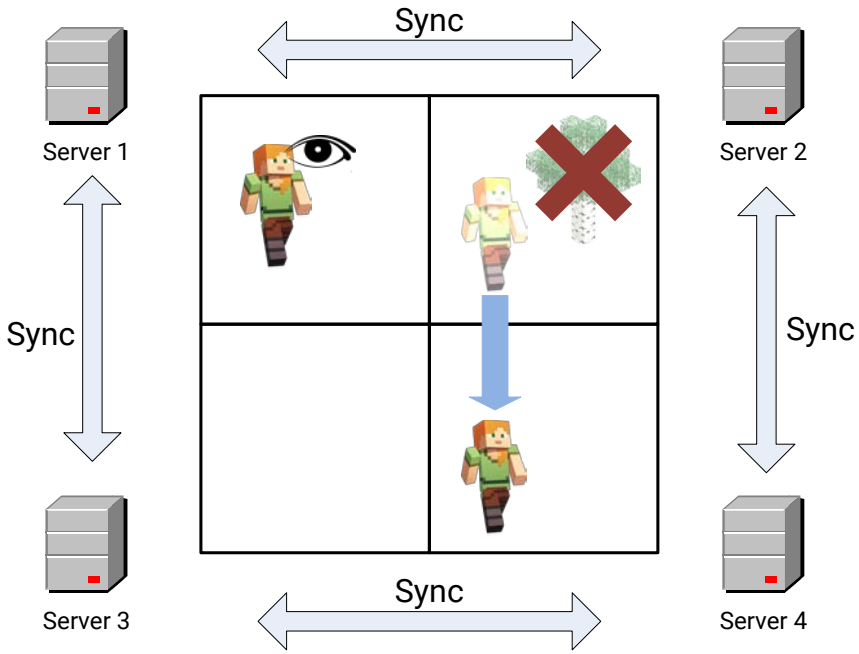
Jeff Burke

UCLA remap

ACM Conference on
Information-Centric Networking 2019

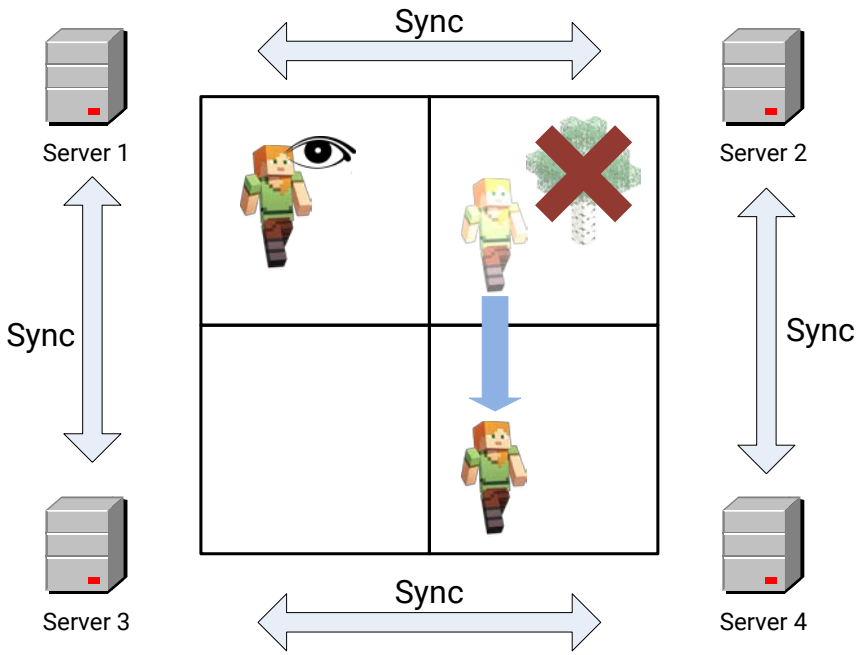
SCALABILITY IN CURRENT ONLINE GAMES

- Computer games are an important factor for the entertainment industry
  - More than 40 billion USD revenues in the US in 2018
  - More than 2.6 billion people played games in 2017
- Up to thousands of players in one game in current massively multiplayer online games
- Simulation of game world usually distributed to server clusters
  - to achieve higher scalability
  - to achieve higher resilience
- *Game state synchronization* is responsible for building a consistent game state across the server cluster

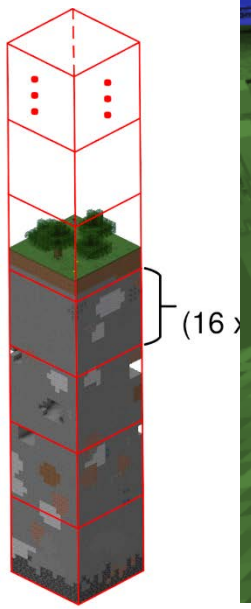


NDN FOR GAME STATE SYNCHRONIZATION

- Benefits of data-oriented architectures
  - Game state synchronization is a data distribution task
  - Point to point communication poorly suited
  - Efficiency increases due to network layer multicast
- Game world well-suited for named access
  - Map regions requested by name instead of from host
  - Separates game world from managing servers
  - Increases scalability and resilience



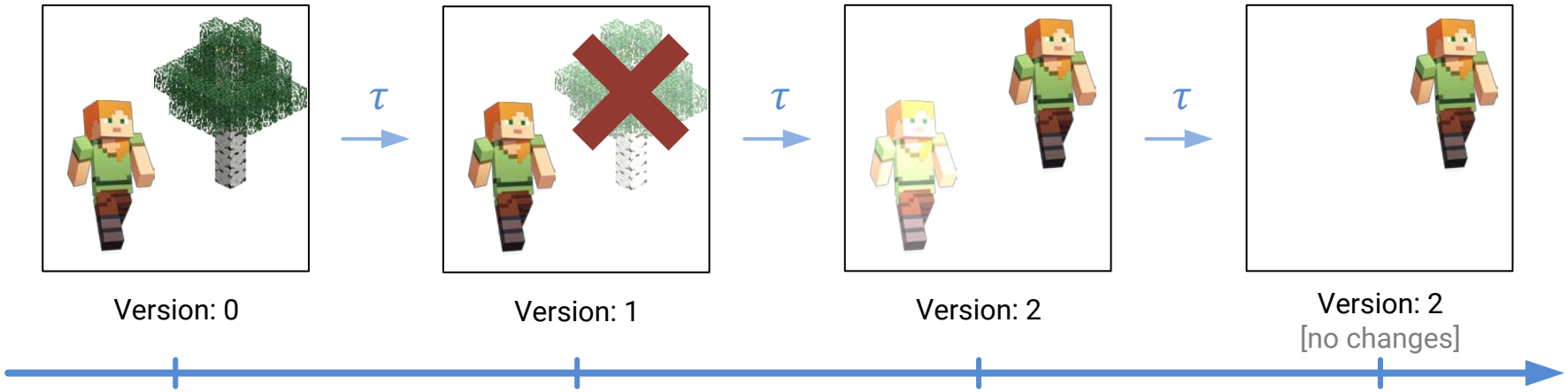
A REAL-WORLD PROTOTYPE WITH MINECRAFT



GAME STATE SYNCHRONIZATION

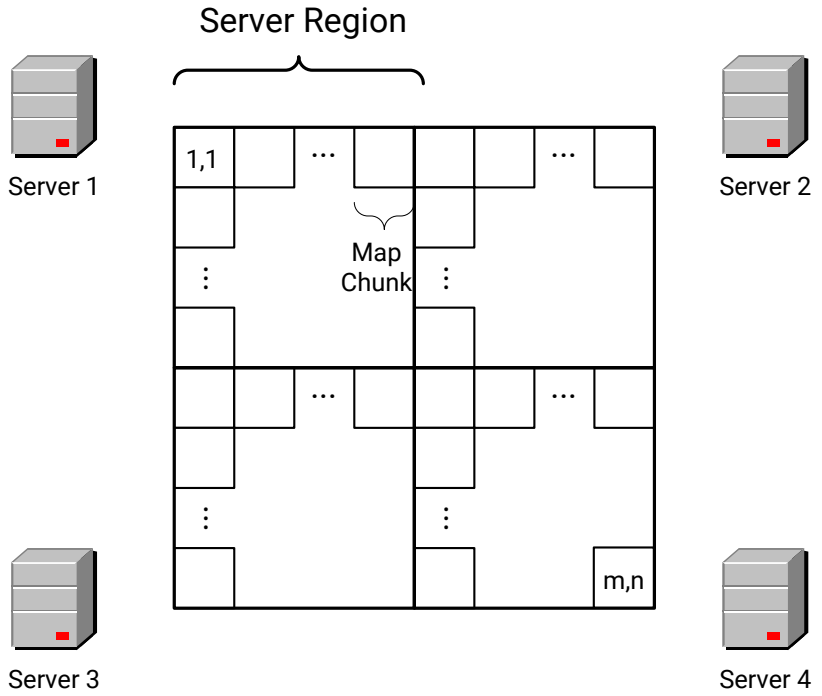
1. Take snapshots of map chunks at discrete time intervals
2. Increase the chunk's version if its state has changed
3. Other servers request the latest version of the chunk's state

How do servers know the latest version of a chunk?



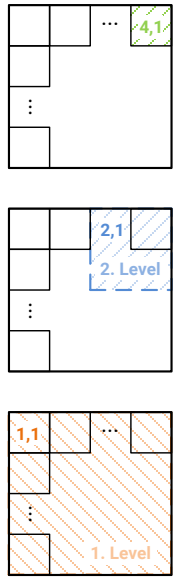
NAMING THE GAME WORLD I

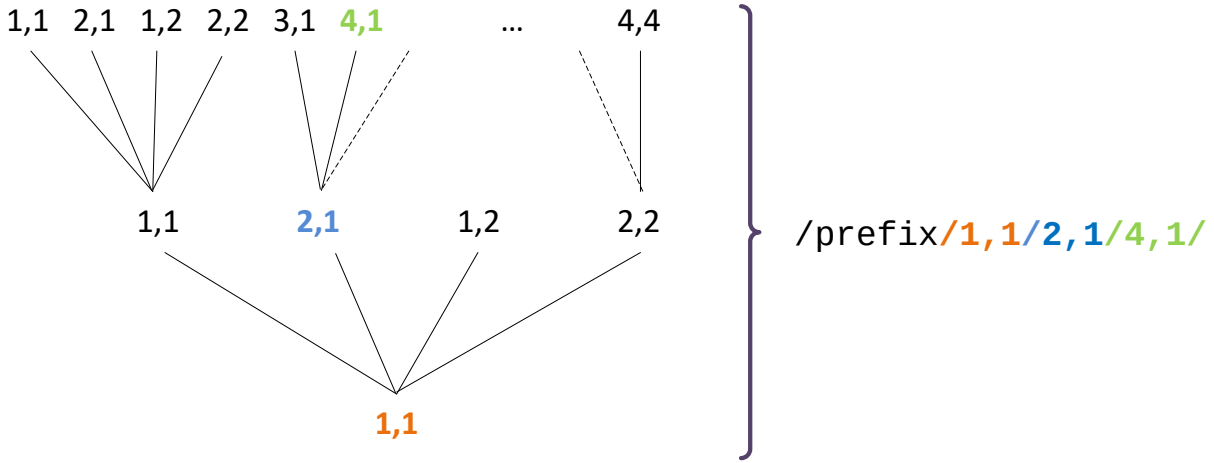
- The game world is divided into map chunks.
- Map chunks contain the full state of small parts of the game world.
- Each server simulates the map chunks of a certain region of the game world (server region).



NAMING THE GAME WORLD II

- Names are created using a quadtree-based hierarchical structure
- Each quadtree level represents a name component
- Keeps the number of required FIB entries low

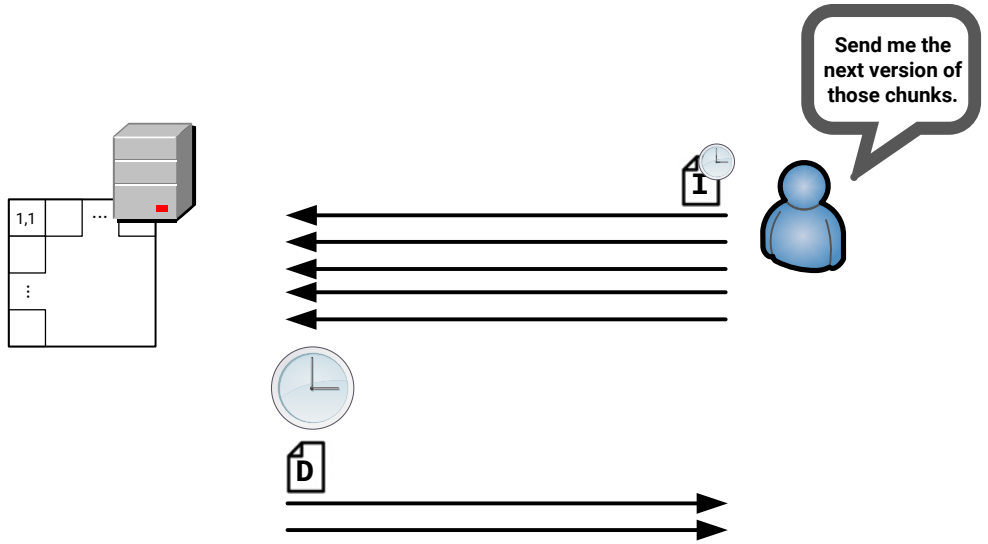




NAÏVE CHUNK-LEVEL UPDATE RETRIEVAL

Chunk-level game state update retrieval

- Issue one Interest for the next version of every world chunk managed by remote servers
- Usage of long-lived Interests instead of polling



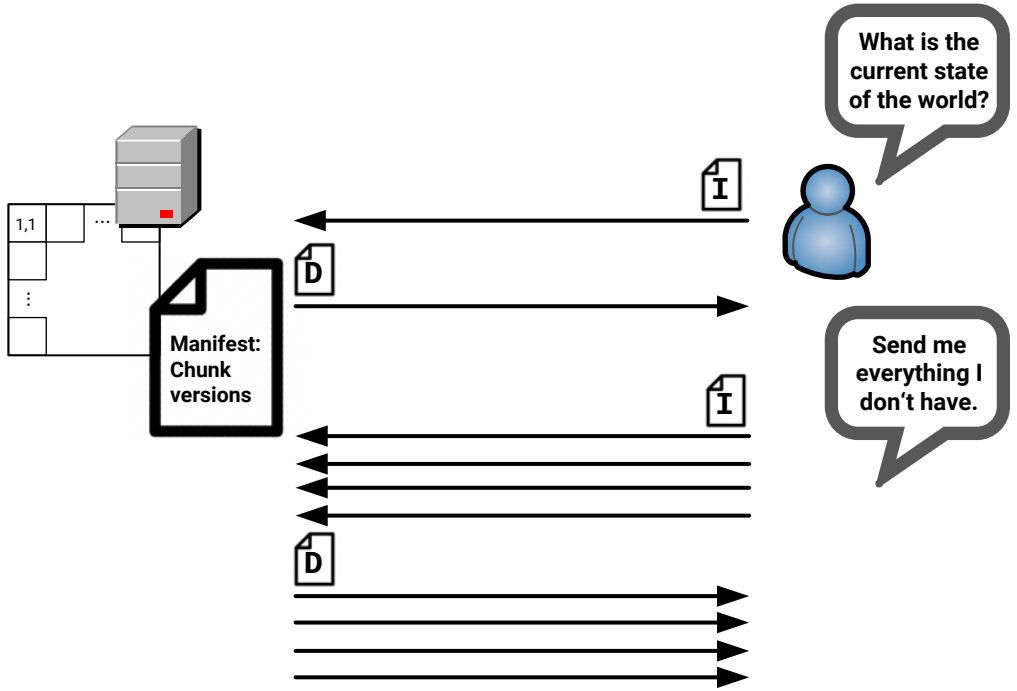
Issues

- Large parts of the world change infrequently
  - Many Interests and PIT entries tend to time out
- Does not scale for large worlds
  - One Interest per world chunk

REGION MANIFEST APPROACH

Region-based update retrieval with manifest files for each server region

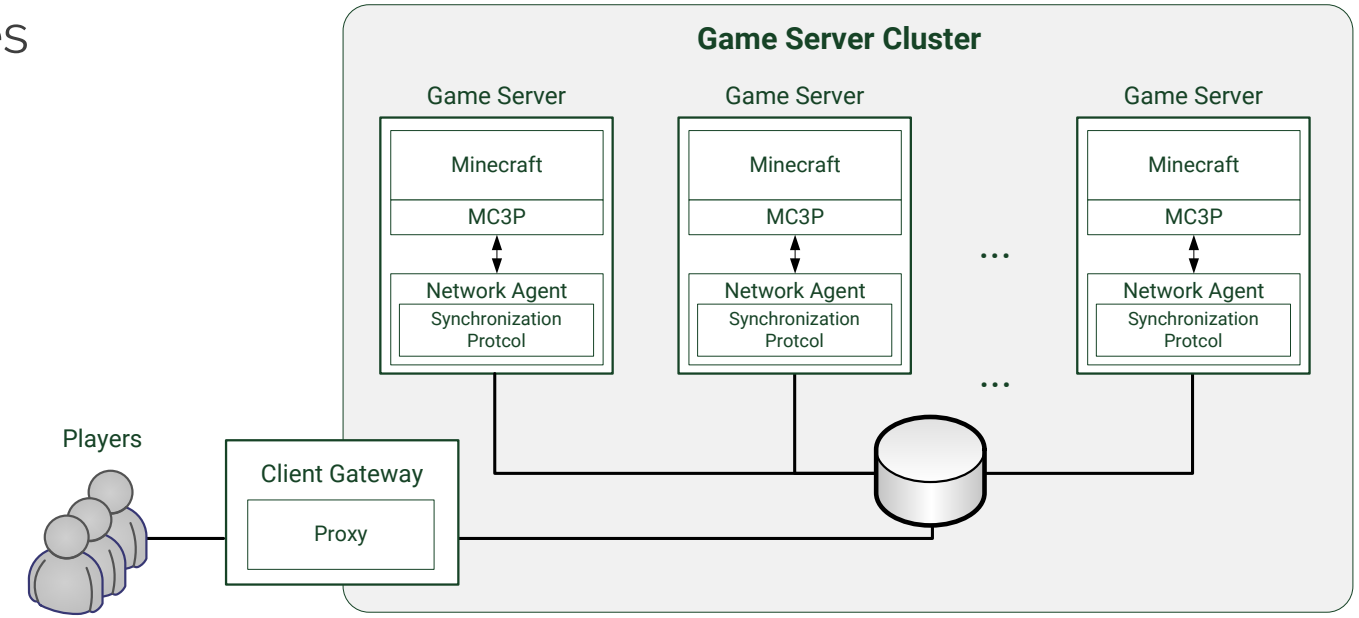
1. Each server provides a manifest file containing chunk versions of its region
2. Other servers request manifest and fetch updates via Interest/Data exchange



- Manifest file optimized for rectangular regions
- Concept borrowed from existing NDN sync protocols

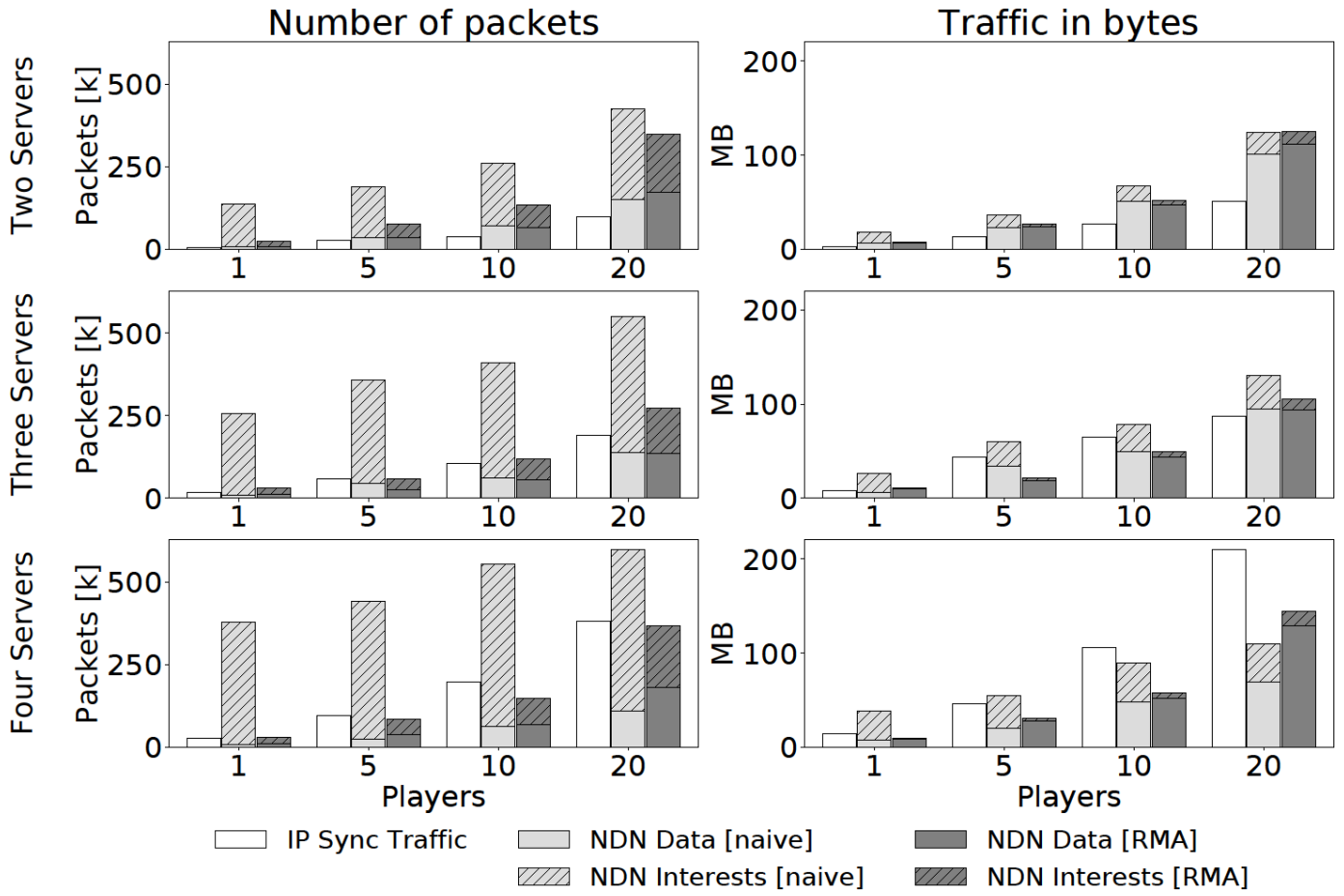
Evaluation Setup

- Multi-server Minecraft prototype
- Server cluster in a local data center
- 1-20 emulated clients playing the game
- Comparison of three sync approaches
  - Baseline IP implementation
  - Naive chunk-level update retrieval
  - Region manifest approach
- MiniNDN as evaluation environment



Evaluation Results

- No benefits in two server setting (due to NDN's protocol overhead)
- Benefits in three and four server scenarios
- Multicast benefits increase in larger server clusters
- RMA approach performs best (three and four server setting)
  - Number of Data packets reduced by more than 50 % (4 server scenario) compared to IP



Conclusion and Future Work

- Use of NDN for game state synchronization
- NDN-based implementation beats IP in terms traffic volume
- Location independent naming decouples map regions from servers

Future work

- increasing scalability of game state synchronization for huge worlds
- usage of NDN for client communication
- consideration of peer-to-peer architectures

Source code available:
https://github.com/phylib/ACM-ICN-19-Reproducibility