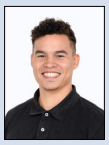

Joël Egger


Marcel Stocker

Diplomanden	Joël Egger, Marcel Stocker
Examinatoren	Prof. Dr. Andreas Rinkel, Lukas Kretschmar
Experte	Knut Schmahl, Lufthansa Industry Solutions Hamburg, Hamburg
Themengebiet	Verschiedenes
Projektpartner	Bundesamt für Bevölkerungsschutz BABS, Bern, BE

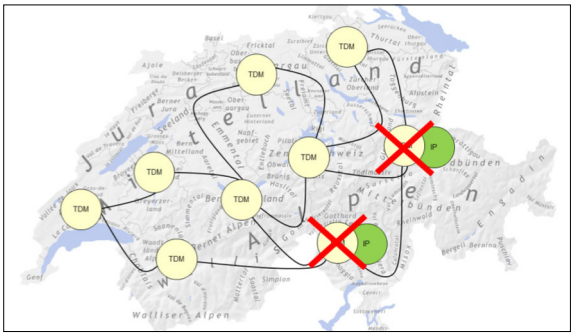
Tetrapolyzer

Sicherstellen der Netzstabilität während der Migration auf die neue Technologie

Organisationsübergreifendes Polycom System
https://blog.alertswiss.ch/de/rubriken/news-babs/polycom-2/

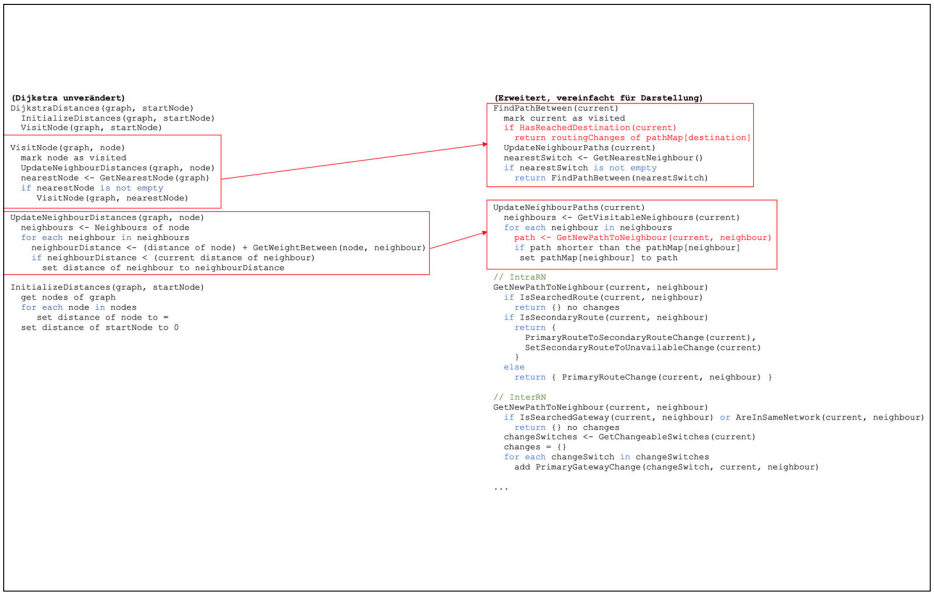
Ausgangslage: Die Blaulichtorganisationen der Schweiz (BORS) nutzen das schweizweit flächendeckende Funksystem POLYCOM. Das POLYCOM-System wird vom Bundesamt für Bevölkerungsschutz (BABS) verwaltet. Dazu wird unter anderem das Simulationsprogramm Tetrapolyzer verwendet. Die aktuell verwendete Technologie des POLYCOM-Systems (TDM-Technologie) ist nun veraltet und muss auf die modernere IP-Technologie migriert werden. Das BABS sieht vor, die Komponenten schrittweise vom alten Netzwerk ins Netzwerk mit der neuen Technologie zu migrieren.

Ziel der Arbeit: Das Ziel dieser Arbeit ist es, während dieser schrittweisen Migration das BABS zu unterstützen, sodass sie die Ausfallsicherheit des alten Netzwerkes immer noch sicherstellen können. Dazu wurde das Simulationsprogramm Tetrapolyzer mit entsprechenden Funktionen erweitert. Mit diesen Funktionen kann das BABS simulieren, welche Konfigurationsänderungen das Entfernen eines oder mehrerer Komponenten aus dem alten Netzwerk mit sich zieht und ob das Netzwerk dabei stabil bleibt.



Migrieren der Komponenten
http://www.wikiwand.com/de/Geographie_der_Schweiz

Ergebnis: Die Problemstellung wurde in die Bereiche Algorithmus (zur Erarbeitung der Konfigurationsänderungen) und Präsentation (Anzeige der Lösungen) unterteilt. Der Algorithmus basiert auf dem Dijkstra-Algorithmus und wurde auf unsere Problemstellung angepasst. Dazu musste der Algorithmus mit mehreren Eigenschaften erweitert werden. Zudem wurde dem Algorithmus eine Abbruchbedingung hinzugefügt, welche eintritt, sobald der gewünschte Ziel-Knoten erreicht wurde. Der Algorithmus wurde mit zwei unterschiedlichen Start-Parametern versehen, sodass zwei unterschiedliche Lösungen gefunden werden. Sobald diese Lösungen bekannt sind, werden die Ergebnisse als Zusammenfassung angezeigt. Details können in einem separaten Fenster angezeigt werden. Das BABS kann die Lösungen vergleichen, eine Entscheidung treffen und die ausgewählte Lösung ins simulierte Netzwerk übernehmen. Danach können die Änderungen ins reale Netzwerk übernommen werden.



```
(Dijkstra unverändert)
DijkstraDistances(graph, startNode)
  InitializeDistances(graph, startNode)
  VisitNode(graph, startNode)

VisitNode(graph, node)
  mark node as visited
  UpdateNeighbourDistances(graph, node)
  nearestNode <- GetNearestNode(graph)
  if nearestNode is not empty
    VisitNode(graph, nearestNode)

UpdateNeighbourDistances(graph, node)
  neighbours <- Neighbours of node
  for each neighbour in neighbours
    neighbourDistance <- (distance of node) + GetWeightBetween(node, neighbour)
    if neighbourDistance < (current distance of neighbour)
      set distance of neighbour to neighbourDistance

InitializeDistances(graph, startNode)
  get nodes of graph
  for each node in nodes
    set distance of node to ∞
  set distance of startNode to 0
```

```
(Erweitert, vereinfacht für Darstellung)
FindPathBetween(current)
  mark current as visited
  if HasReachedDestination(current)
    return routingChanges of pathMap[destination]
  UpdateNeighbourPaths(current)
  nearestSwitch <- GetNearestNeighbour()
  if nearestSwitch is not empty
    return FindPathBetween(nearestSwitch)

UpdateNeighbourPaths(current)
  neighbours <- GetVisitableNeighbours(current)
  for each neighbour in neighbours
    path <- GetNewPathToNeighbour(current, neighbour)
    if path shorter than the pathMap[neighbour]
      set pathMap[neighbour] to path

// IntraRN
GetNewPathToNeighbour(current, neighbour)
  if IsSearchedRoute(current, neighbour)
    return {} no changes
  if IsSecondaryRoute(current, neighbour)
    return {
      PrimaryRouteToSecondaryRouteChange(current),
      SetSecondaryRouteToUnavailableChange(current)
    }
  else
    return { PrimaryRouteChange(current, neighbour) }

// InterRN
GetNewPathToNeighbour(current, neighbour)
  if IsSearchedGateway(current, neighbour) or AreInSameNetwork(current, neighbour)
    return {} no changes
  changeSwitches <- GetChangeableSwitches(current)
  changes = {}
  for each changeSwitch in changeSwitches
    add PrimaryGatewayChange(changeSwitch, current, neighbour)

...
```

Ausschnitte aus dem Pseudocode des Algorithmus
Eigene Darstellung