Resilionator

Endbericht | Call 15 | Projekt ID 5172

Inhalt

1 Einleitung

Im Projekt Resilionator möchten wir ein einfach zu verwendendes Open Source Tool für die Resilienz in Computernetzen bereitstellen. Die Vernetzung ist heutzutage nicht mehr vom alltäglichen Leben wegzudenken. Smartphones und IoT Geräte haben diese Entwicklung in den letzten Jahren vor allem vorangetrieben. Deswegen ist es für Unternehmen, aber auch Haushalte (Smart homes), von großer Bedeutung ihr Netzwerk gegen jegliche Art von Ausfälle abzusichern - resilient zu machen. Denn fällt in einem Netzwerk nur ein kleiner Baustein aus z.B. ein Router oder eine Leitung, kann dies schon das ganze Netzwerk beeinträchtigen und im schlimmsten Fall erhebliche finanzielle Schäden verursachen.

2 Projektbeschreibung

Resilionator richtet sich sowohl an Wissenschaftler, die Open Source Community, aber gerade auch die Industrie und alltägliche Nutzer ohne großes Vorwissen im Bereich der Netzwerke. Aus wissenschaftlicher Sicht können neue Ideen und Algorithmen eingebunden werden, aus Open Source Sicht die funktionalen Komponenten verbessert werden, und aus industrieller Sicht können Schwachstellen erkannt und verbessert, sowie Feedback an die Community gegeben werden. Das Ziel für normale Endnutzer ist es, diesen durch visuelles Feedback einen Einblick in Computernetzwerke und Routingalgorithmen zu gewähren.

3 Verlauf der Arbeitspakete

3.1 Arbeitspaket 1 - *<Detailplanung und Formales am Projektstart>*

Das erste Arbeitspaket beschäftigte sich hauptsächlich mit der formellen Planung des Projekts, wie die Erstellung eines Projektplans, Formulierung der Arbeitspakete, der Inbetriebnahme der Projektwebseite und das Verfassen des ersten Blogeintrags.

3.2 Arbeitspaket 2 - *<Frontend Prototyp>*

In unseren Augen war das zweite Arbeitspaket eines der wichtigsten, denn hier wurden die Entscheidungen über viele Projektgrundlagen getroffen. Angefangen von den Projektspezifikationen (Aussehen und Funktionalität), der Auswahl einer geeigneten

Programmiersprache, Frameworks und Bibliotheken bis hin zur Erstellung des ersten Prototypen und einer anschließenden Aussendung von Einladungen zum Feedback. Eine der wohl wichtigsten Entscheidungen war es die mächtige Python-Library NetworkX in unserem Backend zu benutzen. Diese Entscheidung prägte dann auch die Auswahl der Programmiersprache nämlich Python. Zwar kann man mit Python auch unter anderem Webapps entwickeln, jedoch entschieden wir uns hier für den klassischen Weg einer grafischen Desktop Applikation, welche mit Tkinter der Standard GUI Library von Python umgesetzt wurde. Der Grund hierfür ist, dass eine Desktop GUI Applikation die größte Flexibilität mit kaum Einschränkungen bietet. Python Frameworks für Webapps sind zwar mittlerweile ausgereifter jedoch noch immer mit teilweise größeren Einschränkungen verbunden. Dadurch, dass unsere Applikation "portable" ist d.h. eine Ausführung ist ohne Installation möglich, hat auch der Nutzer kaum Mehraufwand.

3.3 Arbeitspaket 3 - *<Backend Netzaugmentierung>*

Im dritten Arbeitspaket beschrieben wir allererst die formalen Spezifikationen des Backends. Der zweite Schritt war dann, diese Spezifikationen umzusetzen. Wichtig bei diesem Arbeitspaket war eine klare Trennung von Front und Backend. Das Backend ist bei uns für alle Berechnungen zuständig und schickt die Resultate zurück an das Frontend. Das Frontend nimmt Benutzereingaben entgegen, schickt diese dann an das Backend, wartet auf die Resultate und bildet diese grafisch oder textuell ab. Ein großer Punkt bei der Umsetzung des Backends, aber auch Frontends, war die Effizienz. Alle von uns verwendeten Funktionen von NetworkX sind am neuesten Stand der Forschung und deswegen so performant wie es aktuell nur möglich ist z.B. bei der Netzaugmentierung. Im Frontend haben wir eine eigene Garbagecollection eingebaut, welche dafür sorgt, dass GUI Elemente die nicht mehr benötigt werden sofort gelöscht werden bzw. falls möglich wiederverwertet werden. Dies führt zu einer enormen Verbesserung der Ressourcennutzung des Programms.
Schlussendlich haben wir in diesem Arbeitspaket auch die ersten Feedbacks bekommen und auch schon die ersten Verbesserungen durchgeführt. Eine oft genannter Kritikpunkt war, dass es zu umständlich sei Knoten und Kanten immer über das Menü hinzuzufügen. Deswegen haben wir eine "Quickaccess" Leiste implementiert, welche es den Nutzer ermöglicht dies sofort aus der Hauptansicht zu erledigen. Zusätzlich ist es auch möglich das Layout des Graphen zu ändern, denn dies war auch ein großes Anliegen unserer Tester.

3.4 Arbeitspaket 4 - *<Frontend Finalisierung>*

Neben der Finalisierung des Frontends, enthielt dieses Arbeitspaket auch eine Feedbackrunde, bei welcher wir das erste mal potenzielle Endnutzer dieses Produkt vorgestellt haben. Unter

anderem fragten wir Lehrende an der Universität, Studenten der Informatik, aber auch Personen aus anderen Branchen unser Tool zu testen. Dabei war es uns ein großes Anliegen die Installation und Bedienbarkeit des Tools so einfach wie möglich zu gestalten. Deswegen haben wir schon im vorhinein ausführbare Dateien für die gängigsten Betriebssysteme (Windows, Linux, macOS) angefertigt und diese über eine Cloud an die Tester versendet. Diese mussten dann nur noch die jeweils richtige Datei für ihr Betriebssystem herunterladen und ohne jeglichen Installationsprozess ausführen. Nachdem die Feedbacks bei uns eingelangt waren, war es uns wichtig auf jedes einzelne Feedback einzugehen und jeden Verbesserungsvorschlag zu evaluieren. Die Feedbacks der Nutzer halfen uns enorm um die Benutzeroberfläche noch einfacher zu gestalten und Funktionen zu verbessern bzw. zu erweitern.

3.5 Arbeitspaket 5 - *<Backend Routingalgorithmen>*

Dieses Arbeitspaket beinhaltete die Implementierung und Visualisierung des Dijkstra Routing Algorithmus sowie eines Custom-Routingalgorithmus, bei welchem Nutzer den Routingprozess bestimmen können. Der Dijkstra ist einer der bekanntesten und einfachsten Routingalgorithmen, deswegen entschieden wir uns, diesen in unser Tool zu integrieren. Dabei unterscheiden wir zwischen zwei Dijkstra Varianten, nämlich der ursprünglichen Variante bei der Knotenausfälle nach der Berechnung der kürzesten Pfade nicht mehr berücksichtigt werden und einer wo Knotenausfälle berücksichtigt werden. Da es rein lokale Algorithmen sind, kann es aufgrund von Ausfällen zu Schleifen kommen, bei denen das Routing “hängen” bleibt. Das Custom-Routing funktioniert anhand einer Prioritätenliste, in welcher der Nutzer für Knoten seiner Wahl eine bestimmte Knoten angibt, die mit dem Ausgangsknoten benachbart sind. Dies müssen jedoch nicht alle Nachbarn sein. Der Algorithmus bestimmt dann anhand dieser Prioritäten einen geeigneten Pfad. Damit können diverse Routingalgorithmen von extern importiert werden.

3.6 Arbeitspaket 6 - *<Kombination und Dokumentation>*

Dieses Arbeitspaket behandelt die Fertigstellung der nötigen Dokumente zur Projektabgabe. Darunter sind die Fertigstellung unserer Homepage auf Github, welche die auch eine Dokumentation für Nutzer sowohl Entwickler beinhaltet. Desweiteren haben wir noch eine weitere Feedbackrunde abgehalten, bei welcher wir Tester das finale Produkt präsentiert haben. Dies half uns unserem Tool den letzten Feinschliff zu geben sowie Fehler zu beheben.

3.7 Arbeitspaket N - *<Formales>*

Das letzte Arbeitspaket beinhaltet formale Angelegenheiten, welche für den Projektabschluss relevant sind.

4 Umsetzung Förderauflagen

Keine Förderauflagen.

5 Liste Projektendergebnisse

1	***Projektzwischenbericht***	*CC-BY-3.0 AT*	*https://www.netidee.at/resilionator*
2	***Projektendbericht***	*CC-BY-3.0 AT*	*https://www.netidee.at/resilionator*
3	***Entwickler_innen-DOKUMENTATION*** *des Projektergebnisses für andere Entwickler_innen ("Dritte"), die das Projektergebnis nach Projektende nutzen/weiterentwickeln wollen*	*CC-BY-3.0 AT*	*https://www.netidee.at/resilionator*
4	***Anwender_innen-DOKUMENTATION*** *des Projektergebnisses für Anwender_innen, die das Projektergebnis nach Projektende nutzen wollen*	*CC-BY-3.0 AT*	*https://www.netidee.at/resilionator*
5	***Veröffentlichungsfähiger Einseiter***	*CC-BY-3.0 AT*	*https://www.netidee.at/resilionator*
6	*Dokumentation* ***Externkommunikation*** *zur Erreichung Sichtbarkeit /Nachhaltigkeit (Teil des Endberichtes in den Punkten 7-10)*	*CC-BY-3.0 AT*	*https://www.netidee.at/resilionator*
7	***SW-Projektendergebnis***	*BSD-3-Clause*	*https://github.com/philippzabka/resilionator*

6 Verwertung der Projektergebnisse in der Praxis

Unser Projektergebnis, das Tool Resilionator, kann in der Praxis vor allem in der Forschung aber auch in Klein -und Mittelbetrieben und sogar in Haushalten verwendet werden. Forscher bzw. Studenten können mit unserem Tool schnelle Analysen durchführen ohne dabei, wie es oft der Fall ist, jemals selber programmieren zu müssen. KMUs können vor allem die Connectivity und Augmentation Features nutzen um zu überprüfen wie Ausfallsicher ihr Firmennetzwerk tatsächlich ist und welche Maßnahmen sie dagegen unternehmen können um es gegen Knoten- (Router) und Kanten- (Leitungen) ausfälle abzusichern. Zusätzlich eignet sich unser Tool auch für

die Erstellung von Graphen. Diese können in Resilionator schnell und einfach erstellt werden und anschließend für die Weiterverarbeitung in einem gängigen Format exportiert werden.

7 Öffentlichkeitsarbeit/ Vernetzung

Im Rahmen des netidee-Projektes haben wir durch unsere Feedbackrunden schon Interesse in verschiedenen Zielgruppen, national und international, geweckt, und hoffen, dass dieses Interesse durch das finale Tool bzw. den Source Code noch weiter ansteigt. Dieses planen wir dann durch “Word of Mouth” und direktes Anschreiben von Personen aus dem Interessenkreis bzw. über Social Media weiter zu verbreiten. Durch diese stärkere Vernetzung ergibt sich wünschenswerterweise ein “Eigenleben” des Projekts.

8 Eigene Projektwebsite

https://github.com/philippzabka/resilionator

9 Geplante Aktivitäten nach netidee-Projektende

Leider endet die Anstellung von Herrn Zabka mit dem Projektende, ebenso kurz danach die Anstellung von Herrn Förster in Wien. Jedoch ergibt sich eventuell die Möglichkeit, die Weiterentwicklung von Resilionator im Rahmen von weiteren studentischen Arbeiten fortzuführen, z.B. bei Abschlussarbeiten oder im Rahmen von Projekten. Darüber Hinaus möchten wir wie in §7 dargelegt weiter zur Verbreitung und Entwicklung des Tools anregen.

10 Anregungen für Weiterentwicklungen durch Dritte

Resilionator ist ein Tool zur Netzwerkanalyse und die Erweiterungsmöglichkeiten sind deshalb vielfältig. In der Projektlaufzeit konnten wir die geplante Basis an Funktionen implementieren, deswegen bietet es sich an genau hier anzusetzen. Eine direkte Möglichkeit hierbei wäre weitere Routingalgorithmen, welche von NetworkX angeboten werden, in unsere Software zu integrieren. Weiters wäre es interessant zusätzliche Informationen über ein Netzwerk zur Verfügung zu stellen. Einige Beispiele dazu wären: Centrality Measures wie z.B. Betweenness, Degrees (in- und out), Closeness usw.

Weitere Ideen für die Weiterentwicklung wären Quality of Life Features um das Nutzererlebnis so einfach wie Möglich zu gestalten. Ein Beispiel hierfür, welches bereits von uns implementiert ist, ist die Generierung eines zufälligen Graphen per Knopfdruck. Diese Funktion ermöglicht Nutzern

nämlich sofort die Funktionen unseres Tools zu auszuprobieren, ohne den Vorgang der Grapherstellung. Hierbei gibt es sehr viele Arten von Graphen welche erstellt werden, jedoch ist nur eine in Resilionator implementiert. Auch hier bietet NetworkX wieder eine Vielzahl an Funktionen an um solche Graphen zu generieren.

Auch das UI/UX kann stetig verbessert werden. Uns ist weiters aufgefallen, dass die Software auf macOS noch z.T. leichte Anzeigefehler und Performanceprobleme hat. Dies wird leider höchstwahrscheinlich mit der Standard-Bibliothek von Python, Tkinter, zusammenhängen. Unter Linux und Windows sind uns keinerlei solcher Probleme aufgefallen.