Richtlinie zum „Software-Sprint“

Staying.live

Schlussbericht

Zuwendungsempfänger:

Yoshua Wuijts

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IS18S06 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

Kurze Darstellung der Aufgabenstellung und Motivation

Wir haben uns vorgenommen, eine Peer-to-Peer-Lösung für das Video-Streaming im Browser zu entwickeln. Dieses Ziel wollten wir durch die Verwendung des neuartigen Dat-Protokolls erreichen. Der erste Meilenstein, den wir erreichen wollten, war der Umbau bestehender Video-Streaming-Technologien, insbesondere die Erstellung eines neuen HLS-Decoders in Kombination mit der In-Browser-Videoerfassung.

Der nächste Meilenstein bestand darin, die Streaming-Technologien mit dem Dat-Protokoll zu integrieren, um P2P-Video-Streaming zu erstellen. Der letzte Meilenstein bestand in der Vermarktung, Dokumentation und Zusammenarbeit mit Partnern, um die Technologie zu testen.

Beitrag des Projektes zu den Zielen der Förderinitative „Software-Sprint“

Wir haben uns gezielt an Entwickler gewandt, die Anwendungen für Standorte mit schlechter Internetverbindung entwickeln. Wir haben eine Reihe von Protokollen entwickelt, die auf die native Entwicklung abzielen und einige Verbindungsprobleme in diesen Bereichen lösen können. Wir haben das Dat-Protokoll übernommen und seine Streaming-Technologie angepasst, um auf nativen Plattformen mit der Rust-Sprache zu arbeiten. Dies ermöglicht uns, das Hypercore-Streaming-Modul auf mobilen, nativen und IoT-Plattformen zu verwenden.

Ausführliche Darstellung der Ergebnisse

Innerhalb der letzten 6 Monate haben wir 22 separate Repositories auf GitHub veröffentlicht. Das Ziel war es, in Rust eine native Version der Streaming Technologie zu bauen, die dem Dat Protoll zugrunde liegt. Dieses Vorhaben erwies sich als sehr erfolgreich.

Aufgrund ihrer hohen Performanz, Korrektheit und Produktivität war hierfür Rust die Sprache unserer Wahl. Rückblickend sind wir davon überzeugt, damit die richtige Entscheidung getroffen zu haben.

Unter anderem haben wir während der Förderlaufzeit ein abstraktes Persistenz-Backend erstellt, das sowohl Arbeitsspeicher (RAM) als auch Festplattenspeicher (Disk) effizient nutzt. Wir haben auch eine Sammlung von Datenstrukturen (Merkle Tree, Bin-Number, Bitfields) und Parsern (Netzwerk, SLEEP-Header) erstellt.

Während des gesamten Prozesses haben wir beim Erstellen dieser crates (siehe auch crates.io) automatisierte Tests durchgeführt (Property Testing, Fuzzing, Regressionstests, Unit Tests) und sind recht zuversichtlich, dass sie nur wenige Bugs enthalten.

Wir konnten die Grundlagen des Protokolls erfolgreich wiederherstellen, den Video-Streaming-Teil konnten wir nicht komplett umsetzen - daher haben wir uns dagegen entschieden, diesen Teil zur Verwendung freizugeben. Die Technologien, die wir auf dem Weg dorthin gebaut haben können jedoch genutzt werden, um eine zuverlässigere und performantere Implementation unseres Prototypen zu erstellen.

Weitere erwähnenswerte Ergebnisse:

- Wir haben während des gesamten Förderzeitraums Transparenz hergestellt, durch wöchentliche Updates, Live-Coding-Sitzungen [1] und regelmäßige Check-ups mit Mentoren und Beratern.
- Wir haben gelernt, wie man großartige Dokumentation sowohl in Form von Changelogs [2] als auch mithilfe des RustDoc-Tools [3] erstellt.
- Wir haben gelernt, wie man mit jedem Code, der mit branchenführenden Methoden geschrieben wurde, gründliche Tests durchführt.
- Wir haben Projekte in der Sprache Rust erstellt, die bisher über 35.000 Mal heruntergeladen wurden [4].
- Wir sind jetzt Teil der Arbeitsgruppe von Rust Networking, die sich darauf konzentriert, Networking in der Programmiersprache Rust zugänglicher zu machen.

Zielgruppe, Nutzen und mögliche Weiterentwicklungen

Entwickler können nun Teile des Protokolls direkt auf nativen Plattformen verwenden. Alle Core Datenstrukturen für Hypercore sind jetzt implementiert und können verwendet werden.

Wir haben auch einige Contributors mit in das Projekt aufgenommen und standen in direktem Kontakt mit Mitgliedern der wissenschaftlichen Community als auch der Browser-Anbieter.

Bei der Neuimplementierung des Dat-Protokolls fanden wir mehrere Teile, die entweder nicht dokumentiert waren oder in denen wichtige Details fehlten und verbesserten sie.

Für den Zeitraum Oktober bis Dezember 2018 haben wir von Code for Science & Society den Auftrag erhalten, unsere Erfahrungen mit der Protokollimplementierung in Rust zu dokumentieren und einen Leitfaden zu schreiben, um die Implementierung in anderen Sprachen zu reproduzieren. Auch im letzten Teil der Implementierungsphase haben wir große Teile unserer QA-Methodik implementiert und automatisiert, um die langfristige Nachhaltigkeit des Projekts sicherzustellen.

Wir verpflichten uns, das Protokoll in Zukunft weiterzuführen, und arbeiten derzeit an der Finanzierung, um unsere Arbeit 2019 fortzusetzen.

Kurze Darstellung der Arbeiten, die zu keiner Lösung geführt haben

In den früheren Phasen des Projekts hatten wir eine leichte Kursänderung. Nachdem wir uns WebRTC näher angesehen haben, schien unser geplanter Ansatz nicht durchführbar. Wir haben auch festgestellt, dass mindestens ein weiterer Entwickler an ähnlichen Themen gearbeitet hat. Anstatt also tiefer in die Arbeit mit Browsern zu gehen, haben wir uns stattdessen auf Mobilgeräte konzentriert.

In den ersten Wochen haben wir auch mit Video-Streaming-Experten des Video-Dev-Kanals gesprochen und festgestellt, dass das Paket die erforderlichen Methoden zur Kombination mit Video-Streaming über das Dat-Protokoll bietet, sodass kein HLS-Decoder mehr benötigt wird.

Bei der Implementierung des Protokolls in Rust stellten wir fest, dass die async io APIs noch nicht für die Verwendung bereit waren. Daher mussten wir auf die synchronen APIs zurückgreifen. Daraufhin sind wir der Netzwerkarbeitsgruppe beigetreten, um async io zugänglicher zu machen.

Kurze Angabe von Präsentationsmöglichkeiten für mögliche Nutzer

Derzeit gibt es zwei Orte, an denen das Projekt angezeigt werden kann:

- https://github.com/datrs
- https://datrs.yoshuawuyts.com

Kurze Erläuterung zur Einhaltung der Arbeits- und Kostenplanung

Da wir unseren Fokus von Web zu Mobile geändert haben, war eine Neuausrichtung erforderlich. Wir haben eine konkrete Roadmap erstellt, um Hypercore aufzubauen und umzusetzen. Wir begannen mit der Implementierung der Kerndaten und arbeiteten an der Entwicklung von Hypercore: Einer Streaming-P2P-Bibliothek. Wir haben das meiste, was wir aufbauen wollten, erreicht und sind mit den Ergebnissen zufrieden.

Kurze Darstellung von etwaigen Ergebnissen bei anderen Stellen

Es gibt ein paar Projekte, die staying.live ähneln. Vor allem das Hypervision [5] -Projekt zeigte vielversprechende Ergebnisse mit Videostreaming sowohl in Electron als auch im experimentellen Beaker Browser. Das Projekt konnte aufzeigen, dass Video-Streaming über das Dat-Protokoll nicht nur machbar, sondern auch gut zu nutzen ist. Es wies jedoch auch auf die Unzulänglichkeiten der derzeitigen Implementierungen des Dat-Protokolls hin, insbesondere auf mobilen Plattformen. Dieser Umstand spielte auch eine Rolle bei unserer Entscheidung, uns auf die Erstellung einer Version des Protokolls zu konzentrieren, die auch auf Mobilgeräten funktioniert.

Soweit wir wissen, werden derzeit keine weiteren Projekte entwickelt, um das Dat-Protokoll auf mobile Geräte zu bringen. Wir kennen auch keine Projekte, die versuchen, das Dat-Protokoll mit Streaming-Video-Technologie auf dem Handy zu kombinieren.

Verweise

- [1] https://www.youtube.com/playlist?list=PL2F_NKy2ueKN97_6jNeEobHikhW8s5tlL
- [2] https://github.com/datrs/hypercore/blob/master/CHANGELOG.md
- [3] https://docs.rs/hypercore/0.8.0/hypercore/
- [4] https://crates.io/users/yoshuawuyts
- [5] https://louis.center/p2p-hypervision