

☐☐ Der lange Weg zu einem Local-First-Ökosystem

Software zu entwickeln, die sowohl offline als auch online funktioniert – und das ganz ohne Server –, sollte doch eigentlich nicht allzu schwer sein, oder? Ganz so einfach ist es nicht. Das Team des Projekts Reflection aus Jahrgang 01 gibt in diesem Gastbeitrag einen kurzen Überblick über den aktuellen Stand des Local-First-Ökosystems und einige Probleme, über die sie auf dem Weg gestolpert sind.

Auch wenn Local-First nach einem simplen und sinnvollen Prinzip klingt, steckt der Bereich noch in den Kinderschuhen und es gibt derzeit noch nicht viele Local-First-Apps, die man tatsächlich herunterladen und ausprobieren kann. Selbst für die meisten Entwickler*innen ist unklar, welche Hürden noch überwunden werden müssen, um zu einem ausgereiften Local-First-Tech-Stack zu gelangen.

Nach der Entwicklung und Veröffentlichung einer Beta-Version von Reflection, einem Local-First Texteditor, wissen wir nun genauer, welche Arbeit noch vor uns liegt, um Local-First zur Reife zu bringen. In diesem Beitrag möchten wir einen Überblick darüber geben, wo dieser Stack derzeit steht: Was schon funktioniert, was noch zu tun ist, was Entwickler erwarten können und wo Beiträge am dringendsten benötigt werden.

Niemand sonst ist online

Etwas, das wir zwar erwartet hatten, das in der Praxis aber dennoch nervig ist, ist, dass eine Synchronisierung in einem reinen Peer-to-Peer-Netzwerk nur möglich ist, wenn mindestens ein anderer Peer online ist. In einer kleinen Gruppe bedeutet dies, dass Synchronisierungsmöglichkeiten oft aufeinander abgestimmt werden müssen. Für Notizen bei Treffen ist das meist kein Problem weil alle gleichzeitig online sind, Aber selbst in diesem Fall bedeutet es, dass man die Tagesordnung nicht vorab in ein Pad einfügen und davon ausgehen kann, dass andere sie auch sehen können. Man kann auch nicht davon ausgehen, dass ein*e Kolleg*in, die*der das Treffen verpasst hat, die Notizen später lesen kann. Dieses Problem wird als Nebenwirkung der Datenschutzgarantien von p2panda noch verschärft: Jedes Reflection-Pad kann nur mit anderen Peers synchronisiert werden, die das geheime Passwort kennen (d. h. den Einladungscode in Reflection).

Einige unserer Community-Mitglieder haben schnell Workarounds für dieses Problem entwickelt, beispielsweise indem sie einen Computer ständig online lassen und das Dokument geöffnet halten. Das mag das Problem zwar lösen, ist aber mühsam und offensichtlich keine skalierbare Lösung.

Die langfristige Lösung der meisten Local-First-Projekte für dieses Problem sind sogenannte Support-Nodes. Die Idee dahinter ist, dass es einige Nodes gibt, die (fast) immer online sind und Änderungen von Personen abrufen können, die nur sporadisch online sind. Sie können diese dann an andere weiterleiten, die später online gehen. Diese Support-Nodes können auf einem Server in einem Rechenzentrum laufen, aber auch auf einem Raspberry Pi oder einem ungenutzten Laptop im Wohnzimmer.

Die Implementierung von Support-Nodes ist nicht trivial: Sie erfordert Ende-zu-Ende-Verschlüsselung (damit Support-Nodes, die auf einem Server laufen, die Daten und im Idealfall auch die Metadaten nicht lesen können) sowie eine Reihe von UI/UX-Überlegungen, um sicherzustellen, dass sich Support-Nodes innerhalb des App-Kontexts einfach einrichten, konfigurieren und entfernen lassen. Im Idealfall wird daraus eine standardisierte Infrastruktur, die von verschiedenen Local-First-Apps und -Protokollen gemeinsam genutzt werden kann, sodass ein einziger Support-Node für alle Apps verwendet werden könnte.

Bereits existierende Beispiele sind die [Chatmail](#)-Server von DeltaChat, die ausschließlich verschlüsselte E-Mails verarbeiten und diese nach 90 Tagen automatisch löschen, [Keyhive](#) sowie die [Broker von Nextgraph](#). Der Ansatz von p2panda besteht darin, `p2panda-encryption`` zu nutzen, um geheime Verschlüsselungscodes für eine Gruppe von Benutzer*innen zu generieren, mit denen alle Daten, einschließlich aller Metadaten verschlüsselt werden. Diese werden dann in einer einfachen Key-Value-Datenbank auf dem Support-Node gespeichert, von wo aus andere Teilnehmer*innen sie abrufen können.

Als provisorische Kurzzeitleösung erwägen wir, Reflection so zu erweitern, dass es auch in der Kommandozeile läuft, damit man eine Instanz davon auf einem Server ausführen kann. Dies wäre eine einfache Möglichkeit, die Verfügbarkeit sicherzustellen, allerdings mit einem wesentlichen Nachteil: Der Inhalt der Dokumente würde unverschlüsselt auf dem Server gespeichert werden.

Vereinzelte Dokumente

Jedes Reflection-Dokument befindet sich derzeit in einem eigenen Netzwerk aus Peers, was bedeutet, dass Nutzer*innen jedem einzelnen Dokument, das sie synchronisieren und aufrufen möchten, manuell beitreten müssen. Besonders für Projekte, bei denen eine Gruppe von Personen an mehreren Dokumenten zusammenarbeiten muss, ist das sehr umständlich.

Im Reflection-Team haben wir beispielsweise zwei zentrale gemeinsame Dokumente: eines mit unseren Besprechungsnotizen und ein separates für unsere Roadmap. Außerdem erstellen wir regelmäßig Ad-hoc-Dokumente für den Entwurf von Blogbeiträgen, Förderanträgen und ähnlichen Aufgaben. Da all diese Dokumente organisatorisch zusammengehören, ist es sinnvoll, sie stets zusammen zu synchronisieren. Derzeit gibt es jedoch keine Möglichkeit, in all diesen Dokumenten gleichzeitig „online“ zu sein, es sei denn, man öffnet mehrere Fenster der Reflection-App.

Wir möchten zwar den Aufwand für die Synchronisierung von Gruppen gemeinsamer Dokumente verringern, wollen aber nicht jedes einzelne Dokument, an dem wir jemals gearbeitet haben,

ständig synchronisieren. Angesichts des Peer-to-Peer-Charakters der App ist es wichtig, Netzwerkverbindungen nur gezielt herzustellen, um beispielsweise eine versehentliche Offenlegung der IP-Adressen der Teilnehmer*innen zu vermeiden.

Da p2panda sehr flexibel darin ist, wie die Daten zueinander in Relation stehen, wären „Spaces“, die mehrere Dokumente in einem Kontext gruppieren, bereits möglich. Indem mehrere „Sync-Handles“ am Laufen gehalten werden, würde p2panda auch dann im Hintergrund weiter synchronisieren, wenn die Dokumente von der*die Benutzer*in geschlossen sind. Aus Backend-Sicht wäre es somit bereits recht einfach, einen „Shared Folder“-Workflow zu ermöglichen, würde aber auch einiges an Arbeit an der Benutzeroberfläche der App erfordern.

In Kombination mit p2pandas Gruppenverwaltungs- und Verschlüsselungslösung ([p2panda-spaces](#)) könnten wir für jeden „Space“ angemessene Zugriffskontrollen einrichten, Mitglieder hinzufügen und entfernen sowie alle Dokumente ausschließlich für diese Mitglieder verschlüsseln. Dies befindet sich bei p2panda derzeit noch im Versuchsstadium. Daher möchten wir abwarten, bis sich die APIs stabilisiert haben, bevor wir sie in Reflection integrieren.

Letztendlich wäre es schön, dies als Teil eines Systemdienstes oder eines Hintergrund-Daemons zu realisieren, damit wir Dokumente auch dann synchronisieren können, wenn die Reflection-App geschlossen ist. All das würde jedoch eine sorgfältige Risikoanalyse voraussetzen.

Geräte entfernen

Wir haben kürzlich eine Reihe von „Developer Labs“ Treffen für Entwickler*innen von GTK-Apps veranstaltet. Ein Beispiel, das dabei zur Sprache kam, war die Synchronisierung der Liste der Feed-Abonnements in einem RSS-Reader zwischen mehreren Computern. Aus Sicht des Datenmodells ist dies einfach und benötigt kein komplexes „[Conflict-free replicated data type](#)“ (CRDT), aber man braucht eine Möglichkeit, alte Geräte zu entfernen (z. B. wenn man sie verliert).

Mit dem aktuellen Zugriffsmodell, das wir in Reflection verwenden, hat man, sobald man den Einladungscode kennt, vollen Zugriff. Dieser kann von anderen Geräten nicht widerrufen werden. Zwar gibt es Workarounds für dieses Problem, die potenziell auf App-Ebene implementiert werden könnten (z. B. das Dokument ohne das Gerät, das man entfernen möchte, neu zu erstellen), aber eine saubere Lösung erfordert ein ausgefeilteres Zugriffskontrollsystem.

p2panda ist dabei, Lösungen für eine fein abgestufte Zugriffskontrolle einzuführen, die sowohl die direkte Peer-to-Peer-Synchronisation als auch vollständig verschlüsselte Support-Knoten berücksichtigen. Wir hatten ursprünglich geplant, an einer Testintegration in Reflection zu arbeiten, haben diese Arbeit jedoch aus Zeitgründen verschoben. Die Stabilisierungsarbeiten an diesen APIs laufen derzeit noch. Wenn dies also in einer App benötigt wird, empfehlen wir, abzuwarten, bis dieser Prozess abgeschlossen ist.

Ein anderer Ansatz kann darin bestehen, Berechtigungen mit einem Ablaufdatum zu versehen. Ein Beispiel hierfür ist Meadowcap von [Willow](#). Meadowcap unterstützt keine manuelle Widerrufung,

sondern erlaubt nur die Gewährung von Zugriff für ein bestimmtes Zeitfenster, das jedoch verlängert werden kann. [UCAN](#) ist eine ausgefeiltere Version eines Berechtigungssystems und verfügt über eine Widerrufungsfunktion (unter Verwendung von Ketten). All diese Ansätze könnten mit dem p2panda-Stack kombiniert werden.

Lange Join-Codes

Um derzeit in Reflection zusammenzuarbeiten, teilt man einen 64 Zeichen langen alphanumerischen "Beitrittscode". Dieser entspricht unserer „URL“ zu einem Dokument und dient gleichzeitig als Passwort. 64 Zeichen sind zu lang, um sie laut vorzulesen, wenn man persönlich zusammen sitzt oder telefoniert. Das bedeutet, dass man einen Gruppenchat oder einen ähnlichen, anderen Kommunikationsweg benötigt, um ihn zu teilen, was umständlich sein kann.

Eine mögliche Lösung ist, den Beitrittscode über einen QR-Code zu teilen. Das funktioniert gut auf Mobilgeräten, ist aber für den Desktop keine Option.

Eine andere Möglichkeit ist, den Beitritt über kürzere Codes zu ermöglichen, z. B. nur 4 alphanumerische Zeichen wie bei [Toolkitty](#). Dies ist weniger sicher, da diese Codes viel leichter durch Brute-Force-Angriffe erraten werden können. In Kombination mit Zugriffskontrollen oder in Situationen, in denen Dokumente nur in einem lokalen Netzwerk geteilt werden, kann das jedoch akzeptabel sein. Es besteht aber auch die Gefahr von Unstimmigkeiten, wenn zwei Dokumente denselben kurzen Beitrittscode verwenden; dies müsste daher in der Benutzeroberfläche irgendwie berücksichtigt werden.

Privatsphäre der IP-Adresse

Ein grundlegendes Problem bei Peer-to-Peer-Netzwerken besteht darin, dass man sich direkt mit vielen verschiedenen Endgeräten verbindet und nicht mit einem einzigen Server. Das bedeutet, dass alle Teilnehmer*innen des Netzwerks die IP-Adressen der jeweils anderen kennen. Bei der Zusammenarbeit mit einer kleinen Gruppe vertrauenswürdiger Partner*innen an einem Projekt ohne besondere Sicherheitsanforderungen ist das normalerweise kein Problem. Je nach Anwendungsfall kann das Durchsickern von IP-Adressen jedoch gefährlich sein, sodass es keine gute Idee ist, Peer-to-Peer-Apps ohne zusätzliche Sicherheitsmaßnahmen zu nutzen.

Das ist ein wichtiger Grund warum wir momentan nicht empfehlen, Reflection in Situationen zu verwenden, in denen der Einladungscode öffentlich geteilt wird, z. B. in sozialen Medien. Es ist auch der Grund, warum wir es noch nicht für schutzbedürftige Personen, wie z. B. Aktivist*innen, empfehlen würden. IP-Adressen können dazu verwendet werden, Teilnehmer*innen im Netzwerk zu identifizieren und ins Visier zu nehmen (ähnlich wie Anwaltskanzleien, die Personen wegen Torrenting verfolgen), da sie in vielen Fällen zu einem geografischen Standort oder zur Inhaber*in der Internetverbindung zurückverfolgt werden können.

Die naheliegendste Abhilfe für dieses Problem ist die Verwendung einer Anonymisierungstechnologie wie [Tor](#), [I2P](#), [Katzenpost](#) oder [Veilid](#). Wenn der gesamte Datenverkehr anonymisiert ist, ist das Problem mit der IP-Adresse weitgehend hinfällig. Glücklicherweise hat unsere Netzwerkbibliothek [iroh kürzlich Unterstützung dafür hinzugefügt](#), sodass dieses Problem in Zukunft wahrscheinlich weniger relevant sein wird.

Privat arbeiten

Echtzeit-Zusammenarbeit ist zwar großartig, aber manchmal möchte man sich vom „Fishbowl-Effekt“ (dem Wissen, dass andere beobachten, was man tut) abkoppeln und eine Weile alleine arbeiten.

In Reflection ist das bereits heute möglich, indem man die App in den Offline-Modus schaltet. Wenn man jedoch wieder online geht, hat man keine Kontrolle darüber, wie die eigenen Änderungen mit denen der anderen zusammengeführt werden. Das ist nicht immer ein Problem, aber besonders wenn man für längere Zeit offline ist, herrscht Ungewissheit darüber, was passieren wird, wenn man wieder online geht. Was, wenn meine Änderungen keinen Sinn mehr ergeben? Was, wenn die anderen das Dokument komplett umgeschrieben haben, während ich weg war?

Das kam auch in unseren Anwender*innenbefragungen zur Sprache, in denen die Teilnehmer*innen große Unsicherheit hinsichtlich der Zusammenführung nach der Rückkehr in den Online-Modus äußerten. Einige dachten, dass alle ihre Änderungen gelöscht würden, andere vermuteten, dass ihnen die Unterschiede angezeigt würden, aber fast niemand erwartete das tatsächliche Verhalten (nämlich die automatische Zusammenführung mit einem [CRDT](#)).

Um diesen Unsicherheiten entgegenzuwirken, halten wir es für sinnvoller, eine explizite Möglichkeit anzubieten, um eine eigene Kopie des Dokuments abzuzweigen und privat daran zu arbeiten. Dies würde einen klaren „Zusammenführungs“-Vorgang ermöglichen, wenn man mit seinem privaten Entwurf fertig ist und zur Zusammenarbeit in Echtzeit zurückkehren möchte. Es gibt Forschungen in dieser Richtung, wie beispielsweise Ink&Switchs [Upwelling](#), aber bisher keine fertigen Implementierungen, die produktiv eingesetzt werden.

Wer ist „Purple Horse“?

Derzeit ist es in Reflection nicht möglich, die Gerätenamen zu ändern, die automatisch jedem Gerät zugewiesen werden („Purple Horse“, „Green Panda“ usw.). Das hat sich in der Praxis als Problem erwiesen, da es den Nutzer*innen schwerfällt, sich zu merken, welches Tier zu wem gehört.

Wir könnten zwar die Möglichkeit einführen, dass jeder einfach seinen eigenen Namen und Benutzer-Avatar ändern kann. Das würde aber gewisse Risiken mit sich bringen, die wir nicht eingehen wollen, solange wir keinen langfristigen Plan haben, wie wir mit Identitäten umgehen.

Das große Problem bei benutzer*innendefinierten Identitäten, insbesondere in einem Peer-to-Peer-Kontext ohne zentrale Instanz, die Benutzer*innen verifizieren kann, ist Identitätsbetrug. Es ist vergleichsweise einfach, den Namen und den Avatar einer anderen Person zu kopieren, diese im eigenen Konto zu nutzen und sich als diese Person gegenüber deren Kontakten auszugeben.

Die Alternative zu benutzer*innendefinierten Identitäten besteht darin, dass alle ihr eigenes, persönliches Adressbuch mit den Namen all ihrer Kontakte führen. So kann Identitätsdiebstahl verhindert werden, da der sichtbare Name der Person nicht von einem potenziellen Angreifer festgelegt werden kann. Dies wird als „Pet-Name-System“ bezeichnet. Beispiele hierfür sind [Backchannel](#) und [Spritely](#). Der Nachteil dieses Modells ist, dass es für alle Beteiligten sehr aufwendig ist, Adressbücher manuell anzulegen und zu verwalten.

Derzeit gibt es keine umfassende und für alle Fälle gangbare Lösung für dieses Problem, insbesondere bei großen Gruppen. Eine mögliche kurzfristige Lösung für Reflection wäre das Hinzufügen eines Adressbuchs, in dem Nutzer*innen optional Spitznamen für ihre Kontakte registrieren können. Es braucht aber weitere Forschung, um eine Benutzer*innenerfahrung zu entwickeln, die sowohl sicher als auch komfortabel ist.

Um diesen Onboarding-Prozess zu vereinfachen, wäre ein einheitliches Adressbuch hilfreich, das von mehreren Apps gemeinsam genutzt wird. Wir führen derzeit Gespräche darüber, wie eine System-API unter Linux hierfür aussehen könnte. DeltaChat ist ein gutes Beispiel für eine solche „Host-Umgebung“ mit einem einheitlichen Adressbuch (deine Chat-Kontakte), in der Gruppen erstellt werden können, um [WebXDC](#)-Apps auszuführen.

Und nun?

Im Gespräch mit App-Entwickler*innen haben wir festgestellt, dass es für viele, die neu im Bereich „Local-First“ sind, schwierig sein kann, sich einen Überblick darüber zu verschaffen, was tatsächlich noch fehlt, um eine Local-First-App zu entwickeln. Es sind viele verschiedene Technologien beteiligt, von denen einige recht ausgereift sind (z. B. CRDTs), andere jedoch noch experimentell (z. B. Zugriffskontrollen oder Identitätsmanagement).

Wir hoffen, dass dieser Beitrag dabei helfen kann, zu zeigen, was heute bereits umgesetzt werden kann, und vielleicht sogar einige Leute dazu zu begeistern, an den noch fehlenden Aspekten zu arbeiten.

Wenn du Interesse hast, mitzumachen, [schreib uns gerne eine Nachricht](#) oder folge Modal ([@modal@mastodon.design](#)) und p2panda ([@p2panda@autonomous.zone](#)) auf Mastodon.

Revision #1

Created 13 October 2026 12:37:26 by patricia

Updated 13 October 2026 12:39:13 by patricia