net  : Wie macht man ein neues Internet

Bernd Paysan

Forth–Tagung 2014, Bad Vöslau

Outline

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack-orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack-orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack-orientierte, tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack-orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack-orientierte, kakteenzeitige Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Die sieben Lagen der Zwiebel...

Weil sich eine Menge getan hat, möchte ich mal in aller Ruhe erklären, was so in net2o jetzt alles drin ist (und warum's noch nicht ganz fertig ist).
Diese Elemente braucht man für ein Netzwerk (es sind keine Lagen wie bei ISO/OSI, sondern Komponenten):

1. Das physikalische Layer — das ist unterhalb net2o. Im Moment ist das UDP.
2. Path switching mit Paketen, die 2^n bytes groß sind, und in shared memory geschrieben werden
3. Ephemeral key exchange und Signaturen mit Ed25519; symmetrische authentisierte Verschlüsselung+hash+prng mit Keccak
4. Timing driven delay minimizing flow control
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metadaten (DHT)
7. Apps in einer Sandbox zum Anzeigen/Bearbeiten von Daten

Anforderungen

Skalierbar Es muss mit Milliarden Nutzern arbeiten, von denen die Mehrheit nur ein billiges Smartphone hat (und bei „Internet of Things“ auch mit noch kleinerer Hardware)

Einfach zu implementieren Einfach und kompakt, soll auch lokale Busse wie USB und DisplayPort ersetzen können

Einfach einzusetzen Ein vernünftiges Kommunikationssystem aufzusetzen ist mit derzeitiger Internettechnologie so kompliziert, dass die meisten Leute das an Google und Facebook „outsourcen“

Sicher Gegen eine Reihe verschiedener Angriffsszenarien (Sicherheit ist nicht ein Ding)

Multimedia

Migration Muss einen akzeptablen Mikrationspfad bieten

Anforderungen

Skalierbar Es muss mit Milliarden Nutzern arbeiten, von denen die Mehrheit nur ein billiges Smartphone hat (und bei „Internet of Things“ auch mit noch kleinerer Hardware)

Einfach zu implementieren Einfach und kompakt, soll auch lokale Busse wie USB und DisplayPort ersetzen können

Einfach einzusetzen Ein vernünftiges Kommunikationssystem aufzusetzen ist mit derzeitiger Internettechnologie so kompliziert, dass die meisten Leute das an Google und Facebook „outsourcen“

Sicher Gegen eine Reihe verschiedener Angriffsszenarien (Sicherheit ist nicht ein Ding)

Multimedia

Migration Muss einen akzeptablen Mikrationspfad bieten

Anforderungen

Skalierbar Es muss mit Milliarden Nutzern arbeiten, von denen die Mehrheit nur ein billiges Smartphone hat (und bei „Internet of Things“ auch mit noch kleinerer Hardware)

Einfach zu implementieren Einfach und kompakt, soll auch lokale Busse wie USB und DisplayPort ersetzen können

Einfach einzusetzen Ein vernünftiges Kommunikationssystem aufzusetzen ist mit derzeitiger Internettechnologie so kompliziert, dass die meisten Leute das an Google und Facebook „outsourcen“

Sicher Gegen eine Reihe verschiedener Angriffsszenarien (Sicherheit ist nicht ein Ding)

Multimedia

Migration Muss einen akzeptablen Mikrationspfad bieten

Anforderungen

Skalierbar Es muss mit Milliarden Nutzern arbeiten, von denen die Mehrheit nur ein billiges Smartphone hat (und bei „Internet of Things“ auch mit noch kleinerer Hardware)

Einfach zu implementieren Einfach und kompakt, soll auch lokale Busse wie USB und DisplayPort ersetzen können

Einfach einzusetzen Ein vernünftiges Kommunikationssystem aufzusetzen ist mit derzeitiger Internettechnologie so kompliziert, dass die meisten Leute das an Google und Facebook „outsourcen“

Sicher Gegen eine Reihe verschiedener Angriffsszenarien (Sicherheit ist nicht ein Ding)

Multimedia Muss Audio- und Videoströme unterstützen

Migration Muss einen akzeptablen Mikrationspfad bieten

Anforderungen

Skalierbar Es muss mit Milliarden Nutzern arbeiten, von denen die Mehrheit nur ein billiges Smartphone hat (und bei „Internet of Things“ auch mit noch kleinerer Hardware)

Einfach zu implementieren Einfach und kompakt, soll auch lokale Busse wie USB und DisplayPort ersetzen können

Einfach einzusetzen Ein vernünftiges Kommunikationssystem aufzusetzen ist mit derzeitiger Internettechnologie so kompliziert, dass die meisten Leute das an Google und Facebook „outsourcen“

Sicher Gegen eine Reihe verschiedener Angriffsszenarien (Sicherheit ist nicht ein Ding)

Multimedia Echtzeitfähigkeit, QoS/allozierte Bandbreite

Migration Muss einen akzeptablen Mikrationspfad bieten

Anforderungen

Skalierbar Es muss mit Milliarden Nutzern arbeiten, von denen die Mehrheit nur ein billiges Smartphone hat (und bei „Internet of Things“ auch mit noch kleinerer Hardware)

Einfach zu implementieren Einfach und kompakt, soll auch lokale Busse wie USB und DisplayPort ersetzen können

Einfach einzusetzen Ein vernünftiges Kommunikationssystem aufzusetzen ist mit derzeitiger Internettechnologie so kompliziert, dass die meisten Leute das an Google und Facebook „outsourcen“

Sicher Gegen eine Reihe verschiedener Angriffsszenarien (Sicherheit ist nicht ein Ding)

Multimedia Echtzeitfähigkeit, QoS/allozierte Bandbreite

Migration Muss einen akzeptablen Mikrationspfad bieten

Das ist eine Menge Forschung

Mein üblicher Ansatz is:

1. Kucken, was es schon gibt
2. Auswerten und bewerten
3. Schlussfolgern, dass es kaputt ist
4. Regel #1: "If you want it done right, you have to do it yourself"

Manchmal finde ich etwas, was schon funktioniert.

Das ist eine Menge Forschung

Mein üblicher Ansatz is:

1. Kucken, was es schon gibt
2. Auswerten und bewerten
3. Schlussfolgern, dass es kaputt ist
4. Regel #1: "If you want it done right, you have to do it yourself"

Manchmal finde ich etwas, was schon funktioniert.

Das ist eine Menge Forschung

Mein üblicher Ansatz is:

1. Kucken, was es schon gibt
2. Auswerten und bewerten
3. Schlussfolgern, dass es kaputt ist
4. Regel #1: "If you want it done right, you have to do it yourself"

Manchmal finde ich etwas, was schon funktioniert.

Das ist eine Menge Forschung

Mein üblicher Ansatz is:

1. Kucken, was es schon gibt
2. Auswerten und bewerten
3. Schlussfolgern, dass es kaputt ist
4. Regel #1: "If you want it done right, you have to do it yourself"

Manchmal finde ich etwas, was schon funktioniert.

Das ist eine Menge Forschung

Mein üblicher Ansatz is:

1. Kucken, was es schon gibt
2. Auswerten und bewerten
3. Schlussfolgern, dass es kaputt ist
4. Regel #1: “If you want it done right, you have to do it yourself”

Manchmal finde ich etwas, was schon funktioniert.

Das ist eine Menge Forschung

Mein üblicher Ansatz is:

1. Kucken, was es schon gibt
2. Auswerten und bewerten
3. Schlussfolgern, dass es kaputt ist
4. Regel #1: “If you want it done right, you have to do it yourself”

Manchmal finde ich etwas, was schon funktioniert.

Abstraktionen

Vermeide unnötige Abstraktionen

- Netzwerk: Kabel, verbunden mit Switches (hierarchische Sterntopologie)
- Knoten: shared memory — schreiben in entfernte Knoten, lesen nur lokal (und natürlich nur die Bereiche, in die das vereinbart wurde!)
- Trennung zwischen Kommandos und Daten
- Jedes *Ding* ist eine Datei — mit Metadaten (“tags”) in einer Hashtable, *jedermann* ist ein Schlüssel (auch mit Metadaten)
- Event–driven: Kommandos werden verschickt und beantwortet, sie steuern die Abläufe
-

Abstraktionen

Vermeide unnötige Abstraktionen

- Netzwerk: Kabel, verbunden mit Switches (hierarchische Sterntopologie)
- Knoten: shared memory — schreiben in entfernte Knoten, lesen nur lokal (und natürlich nur die Bereiche, in die das vereinbart wurde!)
- Trennung zwischen Kommandos und Daten
- Jedes *Ding* ist eine Datei — mit Metadaten (“tags”) in einer Hashtable, *jedermann* ist ein Schlüssel (auch mit Metadaten)
- Event–driven: Kommandos werden verschickt und beantwortet, sie steuern die Abläufe
-

Abstraktionen

Vermeide unnötige Abstraktionen

- Netzwerk: Kabel, verbunden mit Switches (hierarchische Sterntopologie)
- Knoten: shared memory — schreiben in entfernte Knoten, lesen nur lokal (und natürlich nur die Bereiche, in die das vereinbart wurde!)
- Trennung zwischen Kommandos und Daten
- Jedes *Ding* ist eine Datei — mit Metadaten (“tags”) in einer Hashtable, *jedermann* ist ein Schlüssel (auch mit Metadaten)
- Event–driven: Kommandos werden verschickt und beantwortet, sie steuern die Abläufe
-

Abstraktionen

Vermeide unnötige Abstraktionen

- Netzwerk: Kabel, verbunden mit Switches (hierarchische Sterntopologie)
- Knoten: shared memory — schreiben in entfernte Knoten, lesen nur lokal (und natürlich nur die Bereiche, in die das vereinbart wurde!)
- Trennung zwischen Kommandos und Daten
- Jedes *Ding* ist eine Datei — mit Metadaten (“tags”) in einer Hashtable, *jedermann* ist ein Schlüssel (auch mit Metadaten)
- Event–driven: Kommandos werden verschickt und beantwortet, sie steuern die Abläufe
-

Abstraktionen

Vermeide unnötige Abstraktionen

- Netzwerk: Kabel, verbunden mit Switches (hierarchische Sterntopologie)
- Knoten: shared memory — schreiben in entfernte Knoten, lesen nur lokal (und natürlich nur die Bereiche, in die das vereinbart wurde!)
- Trennung zwischen Kommandos und Daten
- Jedes *Ding* ist eine Datei — mit Metadaten (“tags”) in einer Hashtable, *jedermann* ist ein Schlüssel (auch mit Metadaten)
- Event–driven: Kommandos werden verschickt und beantwortet, sie steuern die Abläufe
- P2P: Alle Knoten sind gleichberechtigt, Daten sind (redundant) verteilt

Pakete switchen, Verbindungen routen

- Switches sind einfacher, schneller und billiger als Router — also will ich nicht jedes Paket einzeln routen
- Routing wird dann beim Verbindungsaufbau gemacht — wir finden heraus, wohin wir die Pakete schicken müssen

Path Switching

- Die ersten n Bits vom Pfad–Feld nehmen und Ziel adressieren
- Den Pfad um n nach links schieben
- Den Rückwärtspfad bit–reverse hinten einfügen

- Der Empfänger dreht die ganze Adresse um, und bekommt so den Rückpfad

Pakete switchen, Verbindungen routen

- Switches sind einfacher, schneller und billiger als Router — also will ich nicht jedes Paket einzeln routen
- Routing wird dann beim Verbindungsaufbau gemacht — wir finden heraus, wohin wir die Pakete schicken müssen

Path Switching

- Die ersten n Bits vom Pfad–Feld nehmen und Ziel adressieren
- Den Pfad um n nach links schieben
- Den Rückwärtspfad bit–reverse hinten einfügen

- Der Empfänger dreht die ganze Adresse um, und bekommt so den Rückpfad

Pakete switchen, Verbindungen routen

- Switches sind einfacher, schneller und billiger als Router — also will ich nicht jedes Paket einzeln routen
- Routing wird dann beim Verbindungsaufbau gemacht — wir finden heraus, wohin wir die Pakete schicken müssen

Path Switching

- Die ersten n Bits vom Pfad–Feld nehmen und Ziel adressieren
- Den Pfad um n nach links schieben
- Den Rückwärtspfad bit–reverse hinten einfügen

- Der Empfänger dreht die ganze Adresse um, und bekommt so den Rückpfad

Pakete switchen, Verbindungen routen

- Switches sind einfacher, schneller und billiger als Router — also will ich nicht jedes Paket einzeln routen
- Routing wird dann beim Verbindungsaufbau gemacht — wir finden heraus, wohin wir die Pakete schicken müssen

Path Switching

- Die ersten n Bits vom Pfad–Feld nehmen und Ziel adressieren
- Den Pfad um n nach links schieben
- Den Rückwärtspfad bit–reverse hinten einfügen

- Der Empfänger dreht die ganze Adresse um, und bekommt so den Rückpfad

Pakete switchen, Verbindungen routen

- Switches sind einfacher, schneller und billiger als Router — also will ich nicht jedes Paket einzeln routen
- Routing wird dann beim Verbindungsaufbau gemacht — wir finden heraus, wohin wir die Pakete schicken müssen

Path Switching

- Die ersten n Bits vom Pfad–Feld nehmen und Ziel adressieren
- Den Pfad um n nach links schieben
- Den Rückwärtspfad bit–reverse hinten einfügen

- Der Empfänger dreht die ganze Adresse um, und bekommt so den Rückpfad

Pakete switchen, Verbindungen routen

- Switches sind einfacher, schneller und billiger als Router — also will ich nicht jedes Paket einzeln routen
- Routing wird dann beim Verbindungsaufbau gemacht — wir finden heraus, wohin wir die Pakete schicken müssen

Path Switching

- Die ersten n Bits vom Pfad–Feld nehmen und Ziel adressieren
- Den Pfad um n nach links schieben
- Den Rückwärtspfad bit–reverse hinten einfügen

- Der Empfänger dreht die ganze Adresse um, und bekommt so den Rückpfad

Paketformat

	Bytes	*Kommentar*
Flags	2	priority, length, flow control flags
Path	16	Internet 1.0 terminology: “address”
Address	8	Adresse im Speicher, ≈port+sequence number
Data	$64 * 2^{0..15}$	bis zu 2MB Paketgröße, genug für die nächsten 40 Jahre
Chksum	16	kryptographische Checksumme

Handover

- Ein typisches Problem mobiler Internetnomaden: Das Gerät wechselt von einem Netz ins andere
- Damit es weitergeht, braucht man ein „Handover“
- Bei net2o ist das trivial: Solange das Paket eine gültige Checksumme hat, wird die Rückaddresse akzeptiert (und ist dann die neue Zieladresse). Natürlich muss der Sender im neuen Netz ein Routing durchführen.
- Damit bei simultanen Handovers die Verbindung nicht verloren geht, gilt: Auf der neuen Adresse senden, auf der alten noch für eine Zeit lang empfangen

Handover

- Ein typisches Problem mobiler Internetnomaden: Das Gerät wechselt von einem Netz ins andere
- Damit es weitergeht, braucht man ein „Handover“
- Bei net2o ist das trivial: Solange das Paket eine gültige Checksumme hat, wird die Rückaddresse akzeptiert (und ist dann die neue Zieladresse). Natürlich muss der Sender im neuen Netz ein Routing durchführen.
- Damit bei simultanen Handovers die Verbindung nicht verloren geht, gilt: Auf der neuen Adresse senden, auf der alten noch für eine Zeit lang empfangen

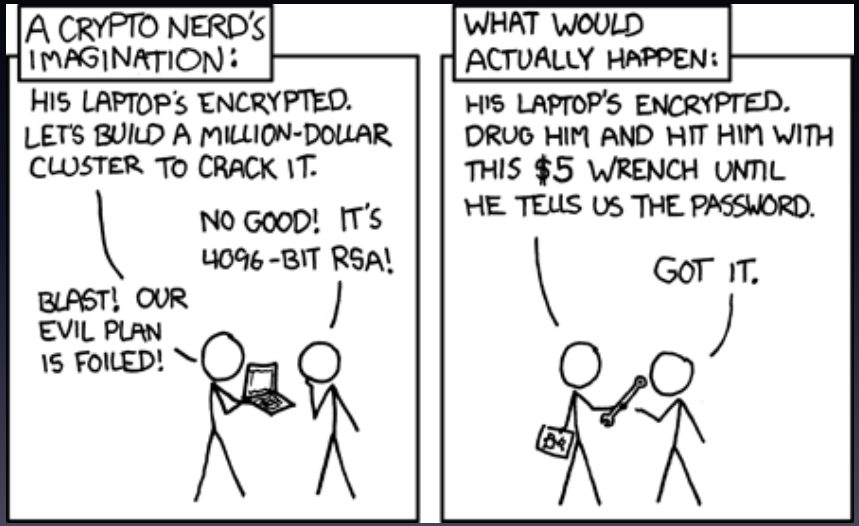
Handover

- Ein typisches Problem mobiler Internetnomaden: Das Gerät wechselt von einem Netz ins andere
- Damit es weitergeht, braucht man ein „Handover“
- Bei net2o ist das trivial: Solange das Paket eine gültige Checksumme hat, wird die Rückaddresse akzeptiert (und ist dann die neue Zieladresse). Natürlich muss der Sender im neuen Netz ein Routing durchführen.
- Damit bei simultanen Handovers die Verbindung nicht verloren geht, gilt: Auf der neuen Adresse senden, auf der alten noch für eine Zeit lang empfangen

Handover

- Ein typisches Problem mobiler Internetnomaden: Das Gerät wechselt von einem Netz ins andere
- Damit es weitergeht, braucht man ein „Handover“
- Bei net2o ist das trivial: Solange das Paket eine gültige Checksumme hat, wird die Rückaddresse akzeptiert (und ist dann die neue Zieladresse). Natürlich muss der Sender im neuen Netz ein Routing durchführen.
- Damit bei simultanen Handovers die Verbindung nicht verloren geht, gilt: Auf der neuen Adresse senden, auf der alten noch für eine Zeit lang empfangen

Sicherheit: Indirekte Attacken sind billiger



Schlüsselaustausch

Welche Algorithmen kann man nehmen?

RSA Schlüssel mit sinnvoller Stärke sollten Richtung 4kBit gehen, und es gibt immer mal wieder kleine Durchbrüche beim Faktorisieren großer Zahlen, die dann die effektive Schlüsselstärke weiter reduzieren. 4kBit sind 512 Byte, das ist zu groß für die 1kB-Pakete, mit denen ich den Schlüsselaustausch mache.

Diffie–Hellman Grundsätzlich gilt da das gleiche Problem, der Algorithmus ist nur etwas anders.

ECC Elliptic Curve Cryptography hat bislang nur ganz generische Attacken, und damit haben 256 Bits immer noch eine Stärke von 128 Bit (also man muss 2^128 komplexe Operationen ausführen und 2^128 Zwischenergebnisse speichern, um den geheimen Schlüssel zu finden)

Curve25519 ist eine elliptische Kurve, die sowohl Signaturen als auch Schlüsselaustausch unterstützt, von den Parametern her sicher ist, und aufgrund der Edwards–Form dem Stand der Technik entspricht.

Schlüsselaustausch

Welche Algorithmen kann man nehmen?

RSA Schlüssel mit sinnvoller Stärke sollten Richtung 4kBit gehen, und es gibt immer mal wieder kleine Durchbrüche beim Faktorisieren großer Zahlen, die dann die effektive Schlüsselstärke weiter reduzieren. 4kBit sind 512 Byte, das ist zu groß für die 1kB-Pakete, mit denen ich den Schlüsselaustausch mache.

Diffie–Hellman Grundsätzlich gilt da das gleiche Problem, der Algorithmus ist nur etwas anders.

ECC Elliptic Curve Cryptography hat bislang nur ganz generische Attacken, und damit haben 256 Bits immer noch eine Stärke von 128 Bit (also man muss 2^128 komplexe Operationen ausführen und 2^128 Zwischenergebnisse speichern, um den geheimen Schlüssel zu finden)

Curve25519 ist eine elliptische Kurve, die sowohl Signaturen als auch Schlüsselaustausch unterstützt, von den Parametern her sicher ist, und aufgrund der Edwards–Form dem Stand der Technik entspricht.

Schlüsselaustausch

Welche Algorithmen kann man nehmen?

RSA Schlüssel mit sinnvoller Stärke sollten Richtung 4kBit gehen, und es gibt immer mal wieder kleine Durchbrüche beim Faktorisieren großer Zahlen, die dann die effektive Schlüsselstärke weiter reduzieren. 4kBit sind 512 Byte, das ist zu groß für die 1kB-Pakete, mit denen ich den Schlüsselaustausch mache.

Diffie–Hellman Grundsätzlich gilt da das gleiche Problem, der Algorithmus ist nur etwas anders.

ECC Elliptic Curve Cryptography hat bislang nur ganz generische Attacken, und damit haben 256 Bits immer noch eine Stärke von 128 Bit (also man muss 2^128 komplexe Operationen ausführen und 2^128 Zwischenergebnisse speichern, um den geheimen Schlüssel zu finden)

Kurve, die sowohl Signaturen als auch Schlüsselaustausch unterstützt, von den Parametern her sicher ist, und aufgrund der Edwards–Form dem Stand der Technik entspricht.

Schlüsselaustausch

Welche Algorithmen kann man nehmen?

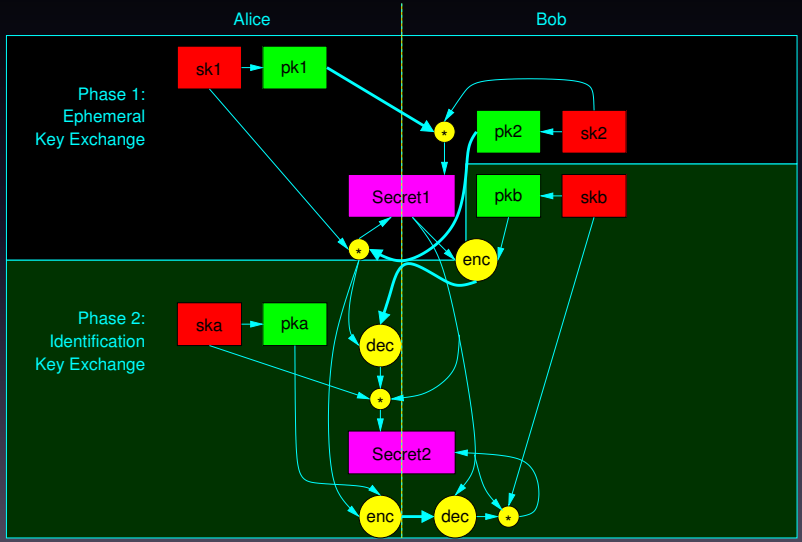
RSA Schlüssel mit sinnvoller Stärke sollten Richtung 4kBit gehen, und es gibt immer mal wieder kleine Durchbrüche beim Faktorisieren großer Zahlen, die dann die effektive Schlüsselstärke weiter reduzieren. 4kBit sind 512 Byte, das ist zu groß für die 1kB-Pakete, mit denen ich den Schlüsselaustausch mache.

Diffie–Hellman Grundsätzlich gilt da das gleiche Problem, der Algorithmus ist nur etwas anders.

ECC Elliptic Curve Cryptography hat bislang nur ganz generische Attacken, und damit haben 256 Bits immer noch eine Stärke von 128 Bit (also man muss 2^128 komplexe Operationen ausführen und 2^128 Zwischenergebnisse speichern, um den geheimen Schlüssel zu finden)

Deshalb ist die Kryptographie der Wahl Dan Bernsteins Ed25519, das ist eine Kurve, die sowohl Signaturen als auch Schlüsselaustausch unterstützt, von den Parametern her sicher ist, und aufgrund der Edwards–Form dem Stand der Technik entspricht.

Ephemeral Schlüsselaustausch+Validation



Herausforderung: Seitenkanalangriffe

- Beim ECC Diffie–Hellman-Schlüsselaustausch ist die Formel $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Alle Operationen, bei der das Geheimnis konstant und der andere Parameter unter Kontrolle des Angreifers sind, können ein Informationsleck bedeuten
- Was man dagegen macht, sind Operationen mit konstanter Zeit und keine datenabhängige Zugriffe; bei Ed25519 berechnet man mit Basis 16 Exponenten vor, und verwendet dann die; das hilft weiter. Die Operationen können aber immer noch unterschiedliche Ströme verbrauchen, und die kann man (sogar akustisch!) immer noch messen
- Phase 1 ist kein Problem, weil jeder Schlüssel nur einmal verwendet wird
- In Phase 2 modifiziere ich den geheimen Schlüssel mit s_1:
  $sk = sk * s_1 \mod l$, $pk = pk * [s_1]$
- Diese Operation $sk * s_1 \mod l$ ist
- Üblich ist für Phase 2 übrigens eine digitale Signatur: der erneute Diffie–Hellman–Schlüsselaustausch ist schneller und versendet weniger Daten.

Herausforderung: Seitenkanalangriffe

- Beim ECC Diffie–Hellman-Schlüsselaustausch ist die Formel $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Alle Operationen, bei der das Geheimnis konstant und der andere Parameter unter Kontrolle des Angreifers sind, können ein Informationsleck bedeuten
- Was man dagegen macht, sind Operationen mit konstanter Zeit und keine datenabhängige Zugriffe; bei Ed25519 berechnet man mit Basis 16 Exponenten vor, und verwendet dann die; das hilft weiter. Die Operationen können aber immer noch unterschiedliche Ströme verbrauchen, und die kann man (sogar akustisch!) immer noch messen
- Phase 1 ist kein Problem, weil jeder Schlüssel nur einmal verwendet wird
- In Phase 2 modifiziere ich den geheimen Schlüssel mit s_1:
  $s_2 = sk * s_1 \mod l$, also $pk_2 = [sk * s_1]$
- Diese Operation $sk * s_1 \mod l$ ist
- Üblich ist für Phase 2 übrigens eine digitale Signatur: der erneute Diffie–Hellman–Schlüsselaustausch ist schneller und versendet weniger Daten.

Herausforderung: Seitenkanalangriffe

- Beim ECC Diffie–Hellman-Schlüsselaustausch ist die Formel $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Alle Operationen, bei der das Geheimnis konstant und der andere Parameter unter Kontrolle des Angreifers sind, können ein Informationsleck bedeuten
- Was man dagegen macht, sind Operationen mit konstanter Zeit und keine datenabhängige Zugriffe; bei Ed25519 berechnet man mit Basis 16 Exponenten vor, und verwendet dann die; das hilft weiter. Die Operationen können aber immer noch unterschiedliche Ströme verbrauchen, und die kann man (sogar akustisch!) immer noch messen
- Phase 1 ist kein Problem, weil jeder Schlüssel nur einmal verwendet wird
- In Phase 2 modifiziere ich den geheimen Schlüssel mit s_1:
- Diese Operation $sk * s_1 \mod l$ ist
- Üblich ist für Phase 2 übrigens eine digitale Signatur: der erneute Diffie–Hellman–Schlüsselaustausch ist schneller und versendet weniger Daten.

Herausforderung: Seitenkanalangriffe

- Beim ECC Diffie–Hellman-Schlüsselaustausch ist die Formel $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Alle Operationen, bei der das Geheimnis konstant und der andere Parameter unter Kontrolle des Angreifers sind, können ein Informationsleck bedeuten
- Was man dagegen macht, sind Operationen mit konstanter Zeit und keine datenabhängige Zugriffe; bei Ed25519 berechnet man mit Basis 16 Exponenten vor, und verwendet dann die; das hilft weiter. Die Operationen können aber immer noch unterschiedliche Ströme verbrauchen, und die kann man (sogar akustisch!) immer noch messen
- Phase 1 ist kein Problem, weil jeder Schlüssel nur einmal verwendet wird
- In Phase 2 modifiziere ich den geheimen Schlüssel mit s_1:
- Diese Operation $sk * s_1 \mod l$ ist
- Üblich ist für Phase 2 übrigens eine digitale Signatur: der erneute Diffie–Hellman–Schlüsselaustausch ist schneller und versendet weniger Daten.

Herausforderung: Seitenkanalangriffe

- Beim ECC Diffie–Hellman-Schlüsselaustausch ist die Formel $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Alle Operationen, bei der das Geheimnis konstant und der andere Parameter unter Kontrolle des Angreifers sind, können ein Informationsleck bedeuten
- Was man dagegen macht, sind Operationen mit konstanter Zeit und keine datenabhängige Zugriffe; bei Ed25519 berechnet man mit Basis 16 Exponenten vor, und verwendet dann die; das hilft weiter. Die Operationen können aber immer noch unterschiedliche Ströme verbrauchen, und die kann man (sogar akustisch!) immer noch messen
- Phase 1 ist kein Problem, weil jeder Schlüssel nur einmal verwendet wird
- In Phase 2 modifiziere ich den geheimen Schlüssel mit s_1: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- Diese Operation $sk * s_1 \mod l$ ist
- Üblich ist für Phase 2 übrigens eine digitale Signatur: der erneute Diffie–Hellman–Schlüsselaustausch ist schneller und versendet weniger Daten.

Herausforderung: Seitenkanalangriffe

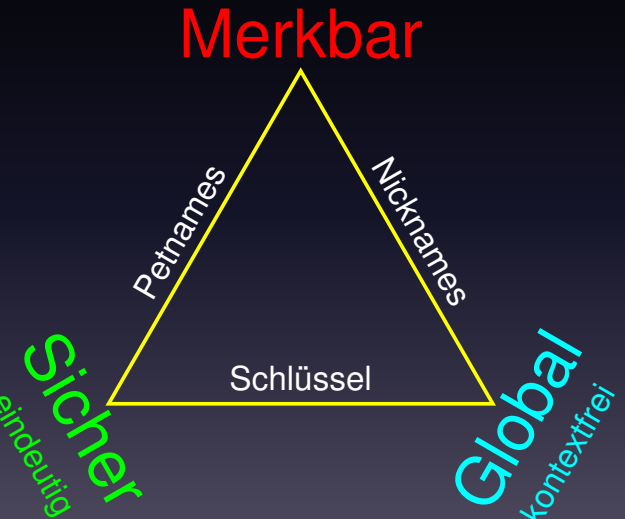
- Beim ECC Diffie–Hellman-Schlüsselaustausch ist die Formel $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Alle Operationen, bei der das Geheimnis konstant und der andere Parameter unter Kontrolle des Angreifers sind, können ein Informationsleck bedeuten
- Was man dagegen macht, sind Operationen mit konstanter Zeit und keine datenabhängige Zugriffe; bei Ed25519 berechnet man mit Basis 16 Exponenten vor, und verwendet dann die; das hilft weiter. Die Operationen können aber immer noch unterschiedliche Ströme verbrauchen, und die kann man (sogar akustisch!) immer noch messen
- Phase 1 ist kein Problem, weil jeder Schlüssel nur einmal verwendet wird
- In Phase 2 modifiziere ich den geheimen Schlüssel mit s_1: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- Diese Operation $sk * s_1 \mod l$ ist
- Üblich ist für Phase 2 übrigens eine digitale Signatur: der erneute Diffie–Hellman–Schlüsselaustausch ist schneller und versendet weniger Daten.

Herausforderung: Seitenkanalangriffe

- Beim ECC Diffie–Hellman-Schlüsselaustausch ist die Formel $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Alle Operationen, bei der das Geheimnis konstant und der andere Parameter unter Kontrolle des Angreifers sind, können ein Informationsleck bedeuten
- Was man dagegen macht, sind Operationen mit konstanter Zeit und keine datenabhängige Zugriffe; bei Ed25519 berechnet man mit Basis 16 Exponenten vor, und verwendet dann die; das hilft weiter. Die Operationen können aber immer noch unterschiedliche Ströme verbrauchen, und die kann man (sogar akustisch!) immer noch messen
- Phase 1 ist kein Problem, weil jeder Schlüssel nur einmal verwendet wird
- In Phase 2 modifiziere ich den geheimen Schlüssel mit s_1: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- Diese Operation $sk * s_1 \mod l$ ist
- Üblich ist für Phase 2 übrigens eine digitale Signatur: der erneute Diffie–Hellman–Schlüsselaustausch ist schneller und versendet weniger Daten.

Trust&PKI: Zookos Dreieck

(Ein Teil der) Herausforderung



Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Trust&PKI

- Certification Authorities (SSL) funktionieren nicht: Ein faules Ei, und das System ist kaputt
- Die viel einfachere Strategie von SSH (Schlüssel merken) funktioniert viel besser
- Damit erfordert nur der erste Kontakt besondere Aufmerksamkeit
- Bei sozialen Netzwerken hat man Freunde von Freunden: Das sind sichere Petnames für sichere Schlüssel
- Die Pubkeys müssen nicht öffentlich sein, um eindeutig zu sein
- Pubkey–Authentication ist viel besser als Passwörter
- In der Regel hat man kein gemeinsames Geheimnis, um z.B. das Socialist Millionaire Protocol zu nutzen
- Das Zooko–Dreieck behandelt das Problem korrekter Zuordnung nicht (Cybersquatter, die sich schon in RL vergebene Namen krallen)

Symmetrische Kryptographie

Auswahl des Algorithmus

An Anfang war der Wurstkessel...

- AEAD–Verschlüsselung: Authentisierte Ver- und Entschlüsselung
- Üblicherweise benutzt: AES in GCM (Galois Counter–Mode)
- Vorsicht: Ein Galois–Feld über 2^n ist kein sicherer Hash, sondern nur eine Polynom–Checksumme, bekannt fragil [2], und die Sicherheit ist ≤ 64 Bits [3], das Paper schlägt vor, GF(p) with $p = 2^{128} + 12451$ zu nutzen
- AES benutzt einen konstanten Key, und macht damit Seitenkanalangriffe einfach
- Ein Counter–Mode ist eigentlich ein Stream–Cipher, also könnte man auch so einen benutzen
- Dann käme DAN BERNSTEIN's xsalsa+poly1305 in Frage: Das ist ein Stream–Cipher und es nutzt ein GF(p) als Prüfsumme. Die ist aber auch nur sicher, wenn die Verschlüsselung sicher ist.

Symmetrische Kryptographie

Auswahl des Algorithmus

An Anfang war der Wurstkessel…

- AEAD–Verschlüsselung: Authentisierte Ver- und Entschlüsselung
- Üblicherweise benutzt: AES in GCM (Galois Counter–Mode)
- Vorsicht: Ein Galois–Feld über 2^n ist kein sicherer Hash, sondern nur eine Polynom–Checksumme, bekannt fragil [2], und die Sicherheit ist ≤ 64 Bits [3], das Paper schlägt vor, GF(p) with $p = 2^{128} + 12451$ zu nutzen
- AES benutzt einen konstanten Key, und macht damit Seitenkanalangriffe einfach
- Ein Counter–Mode ist eigentlich ein Stream–Cipher, also könnte man auch so einen benutzen
- Dann käme Dan Bernstein's xsalsa+poly1305 in Frage: Das ist ein Stream–Cipher und es nutzt ein GF(p) als Prüfsumme. Die ist aber auch nur sicher, wenn die Verschlüsselung sicher ist.

Symmetrische Kryptographie

Auswahl des Algorithmus

An Anfang war der Wurstkessel...

- AEAD–Verschlüsselung: Authentisierte Ver- und Entschlüsselung
- Üblicherweise benutzt: AES in GCM (Galois Counter–Mode)
- Vorsicht: Ein Galois–Feld über 2^n ist kein sicherer Hash, sondern nur eine Polynom–Checksumme, bekannt fragil [2], und die Sicherheit ist ≤ 64 Bits [3], das Paper schlägt vor, GF(p) with $p = 2^{128} + 12451$ zu nutzen
- AES benutzt einen konstanten Key, und macht damit Seitenkanalangriffe einfach
- Ein Counter–Mode ist eigentlich ein Stream–Cipher, also könnte man auch so einen benutzen
- Somit kann man [...] einen sicheren Stream–Cipher und es nutzt ein GF(p) als Prüfsumme. Die ist aber auch nur sicher, wenn die Verschlüsselung sicher ist.

Symmetrische Kryptographie

Auswahl des Algorithmus

An Anfang war der Wurstkessel...

- AEAD–Verschlüsselung: Authentisierte Ver- und Entschlüsselung
- Üblicherweise benutzt: AES in GCM (Galois Counter–Mode)
- Vorsicht: Ein Galois–Feld über 2^n ist kein sicherer Hash, sondern nur eine Polynom–Checksumme, bekannt fragil [2], und die Sicherheit ist ≤ 64 Bits [3], das Paper schlägt vor, GF(p) with $p = 2^{128} + 12451$ zu nutzen
- AES benutzt einen konstanten Key, und macht damit Seitenkanalangriffe einfach
- Ein Counter–Mode ist eigentlich ein Stream–Cipher, also könnte man auch so einen benutzen
- Dann käme Dan Bernstein's xsalsa+poly1305 in Frage: Das ist ein Stream–Cipher und es nutzt ein GF(p) als Prüfsumme. Die ist aber auch nur sicher, wenn die Verschlüsselung sicher ist.

Symmetrische Kryptographie

Auswahl des Algorithmus

An Anfang war der Wurstkessel...

- AEAD–Verschlüsselung: Authentisierte Ver- und Entschlüsselung
- Üblicherweise benutzt: AES in GCM (Galois Counter–Mode)
- Vorsicht: Ein Galois–Feld über 2^n ist kein sicherer Hash, sondern nur eine Polynom–Checksumme, bekannt fragil [2], und die Sicherheit ist ≤ 64 Bits [3], das Paper schlägt vor, GF(p) with $p = 2^{128} + 12451$ zu nutzen
- AES benutzt einen konstanten Key, und macht damit Seitenkanalangriffe einfach
- Ein Counter–Mode ist eigentlich ein Stream–Cipher, also könnte man auch so einen benutzen
- Dann käme DAN BERNSTEIN's xsalsa+poly1305 in Frage: Das ist ein Stream–Cipher und es nutzt ein GF(p) als Prüfsumme. Die ist aber auch nur sicher, wenn die Verschlüsselung sicher ist.

Symmetrische Kryptographie

Auswahl des Algorithmus

An Anfang war der Wurstkessel...

- AEAD–Verschlüsselung: Authentisierte Ver- und Entschlüsselung
- Üblicherweise benutzt: AES in GCM (Galois Counter–Mode)
- Vorsicht: Ein Galois–Feld über 2^n ist kein sicherer Hash, sondern nur eine Polynom–Checksumme, bekannt fragil [2], und die Sicherheit ist ≤ 64 Bits [3], das Paper schlägt vor, GF(p) with $p = 2^{128} + 12451$ zu nutzen
- AES benutzt einen konstanten Key, und macht damit Seitenkanalangriffe einfach
- Ein Counter–Mode ist eigentlich ein Stream–Cipher, also könnte man auch so einen benutzen
- Dann käme DAN BERNSTEIN's xsalsa+poly1305 in Frage: Das ist ein Stream–Cipher und es nutzt ein GF(p) als Prüfsumme. Die ist aber auch nur sicher, wenn die Verschlüsselung sicher ist.

Symmetrische Kryptographie

Auswahl des Algorithmus

An Anfang war der Wurstkessel...

- AEAD–Verschlüsselung: Authentisierte Ver- und Entschlüsselung
- Üblicherweise benutzt: AES in GCM (Galois Counter–Mode)
- Vorsicht: Ein Galois–Feld über 2^n ist kein sicherer Hash, sondern nur eine Polynom–Checksumme, bekannt fragil [2], und die Sicherheit ist ≤ 64 Bits [3], das Paper schlägt vor, GF(p) with $p = 2^{128} + 12451$ zu nutzen
- AES benutzt einen konstanten Key, und macht damit Seitenkanalangriffe einfach
- Ein Counter–Mode ist eigentlich ein Stream–Cipher, also könnte man auch so einen benutzen
- Dann käme Dan Bernstein's xsalsa+poly1305 in Frage: Das ist ein Stream–Cipher und es nutzt ein GF(p) als Prüfsumme. Die ist aber auch nur sicher, wenn die Verschlüsselung sicher ist.

Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist >256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH

  Vorteil: Die, die der NSA vertrauen, können der NSA vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist >256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- Keccak erfüllt alle Bedingungen an Vertrauenswürdigkeit: Die, die dem NIST vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist >256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- Keccak erfüllt alle Bedingungen an Vertrauenswürdigkeit: Die, die dem NIST vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist $>$256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- Keccak erfüllt alle Bedingungen an Vertrauenswürdigkeit: Die, die dem NIST vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist >256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- Die, die der NSA vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist >256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- Ich verwende Keccak, weil ich der Kryptographie–Community mehr vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist >256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

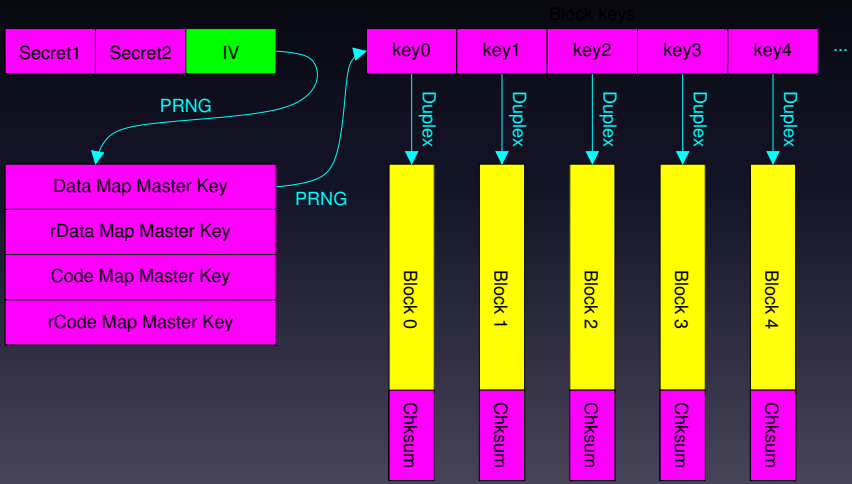
Keccak

Hat die Features von Wurstkessel, ist aber aus einem NIST–Wettbewerb als Sieger hervorgegangen

- Vorschlag: Einen starken Hash für die Autentisierung
- Der offensichtliche Kandidat ist der SHA–3–Gewinner, Keccak, der eine sehr gute Kryptanalyse hinter sich hat
- Noch besser: Wie Wurstkessel kann Keccak im Duplex–Mode auch ver- und entschlüsseln (praktisch ohne zusätzliche Kosten)
- Es gibt keinen konstanten Key, also perfekte Sicherheit gegen Seitenkanalangriffe
- Die Stärke ist >256 Bits, während AES–256 mit Related–Key–Attacken zu kämpfen hat, also effektiv deutlich schwächer ist
- Keccak ist ein universelles Krypto–Primitiv, während man bei AES noch zwei weitere braucht: hash+AES+GHASH
- Keccak erfüllt alle Bedingungen an Vertauenswürdigkeit: Die, die dem NIST vertrauen, weil es vom NIST ist, und die, die der NSA misstrauen, weil es aus einem Wettbewerb hervorgegangen ist. I verwende Keccak mit $r = 1024$ und Kapazität $c = 576$, so wie von den Autoren ursprünglich vorgeschlagen

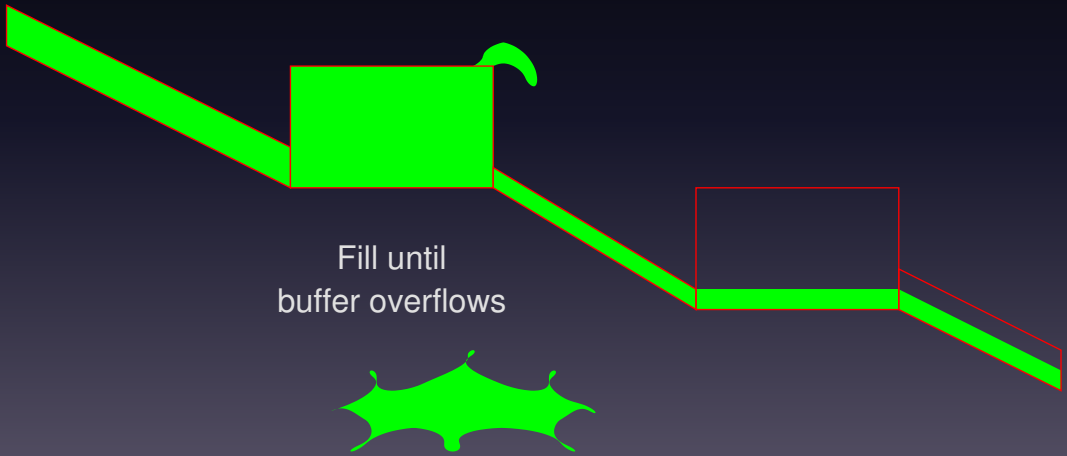
Schlüsselverwendung

Alle Schlüssel dürfen nur einmal verwendet werden!



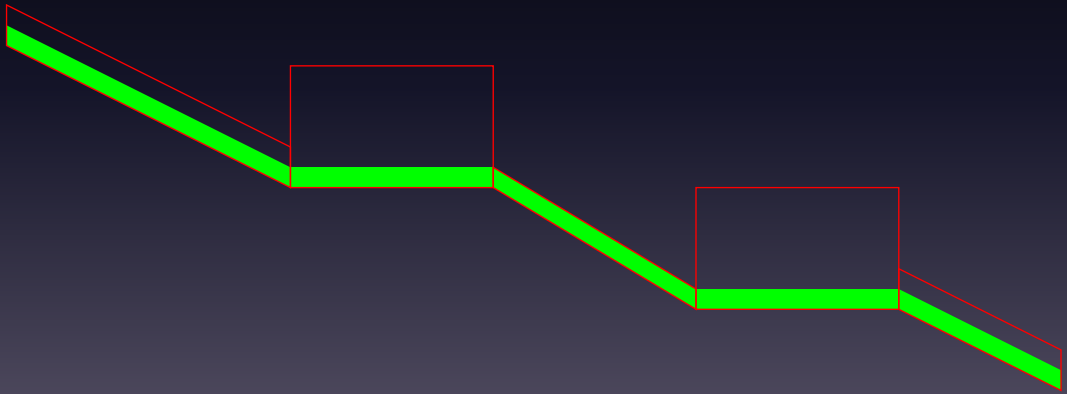
Flusskontrolle (kaputt)

- TCP füllt einfach den Puffer auf, bis ein Paket verloren geht, statt die Rate vorher schon zu reduzieren. Name des Symptoms: “Buffer bloat”. Allerdings sind Puffer essentiell für gute Netzwerkperformance.



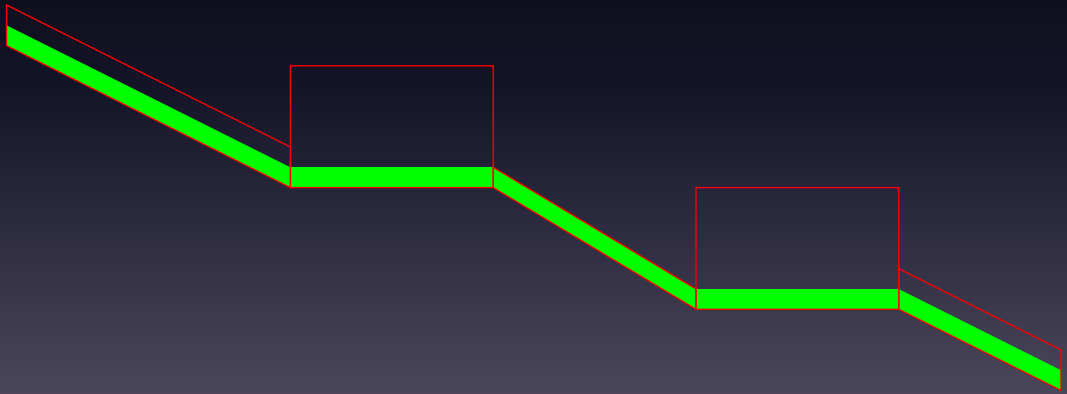
Alternativen?

- LEDBAT versucht, einen konstanten, kleinen Delay zu erzeugen: Geht so-lala, ist aber unfair (der letzte gewinnt)
- CurveCPs Flusskontrolle ist „eine Menge Forschung,“ haha (es ist nichts da)
- Also musste was neues her



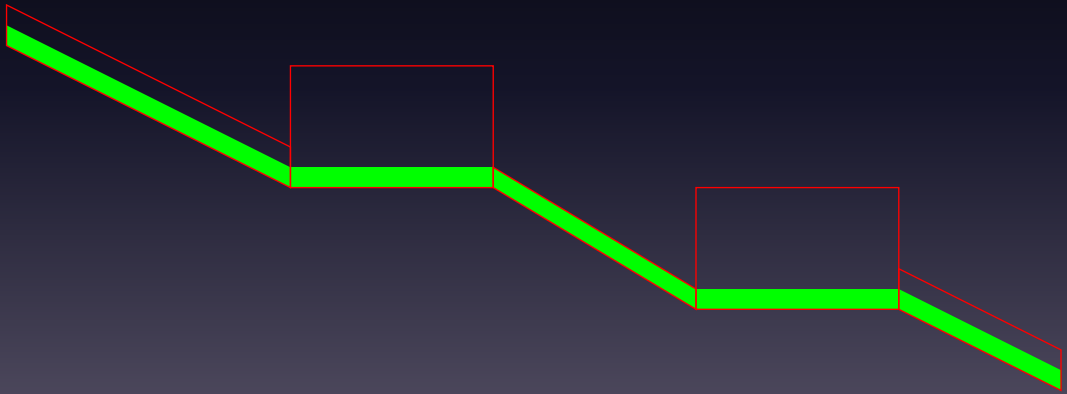
Alternativen?

- LEDBAT versucht, einen konstanten, kleinen Delay zu erzeugen: Geht so-lala, ist aber unfair (der letzte gewinnt)
- CurveCPs Flusskontrolle ist „eine Menge Forschung,“ haha (es ist nichts da)
- Also musste was neues her



Alternativen?

- LEDBAT versucht, einen konstanten, kleinen Delay zu erzeugen: Geht so-lala, ist aber unfair (der letzte gewinnt)
- CurveCPs Flusskontrolle ist „eine Menge Forschung,“ haha (es ist nichts da)
- Also musste was neues her



net2o–Flusskontrolle

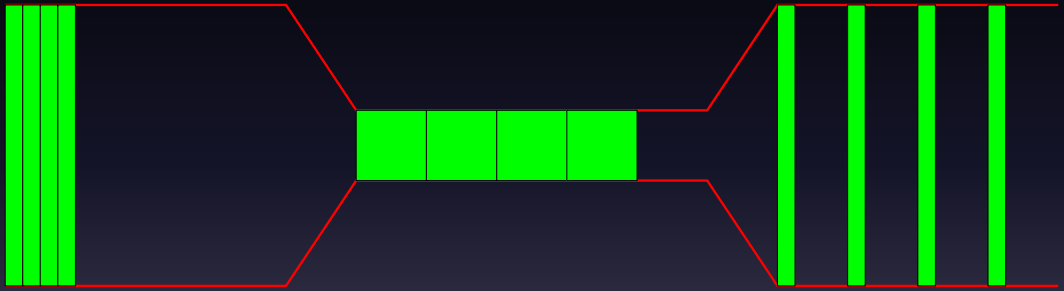


Abbildung : Miss den Flaschenhals mit einem Burst

Client misst, Server setzt die Rate

Client zeichnet die *Zeitdifferenz* für das erste und letzte Paket in einen Burst, und extrapoliert daraus die mögliche Rate *rate*.

$$rate := \Delta t * \frac{burstlen}{packets - 1}$$

Server würde einfach diese Rate nutzen

Client misst, Server setzt die Rate

Client zeichnet die *Zeitdifferenz* für das erste und letzte Paket in einen Burst, und extrapoliert daraus die mögliche Rate *rate*.

$$rate := \Delta t * \frac{burstlen}{packets - 1}$$

Server würde einfach diese Rate nutzen

Fairness

Fairness bedeutet, dass gleichzeitige Verbindungen in etwa die gleiche Datenrate bekommen, sich also die Leitung fair teilen.

- Idealerweise, der Router/Switch würde Pakete verschiedener Verbindungen round–robin senden. Das würde die berechnete Bandbreite entsprechend korrigieren, und wäre auch eine große Erleichterung beim TCP–Buffer–Bloat–Symptom, weil jede Verbindung ihren eigenen Puffer auffüllen würde.
- Der Vorschlag „Fair Queuing“ ist von 1985, und es gibt Router, die das auch implementieren (als konfigurierbare Option) — aber leider z.B. nicht für DSLAMs, die haben FIFO–Queuing
- Daraus ergibt sich die Notwendigkeit, auch für FIFO–Queuing einen stabilen Algorithmus zu finden.
- Der Algorithmus muss sich geänderten Bedingungen schnell anpassen. Insbesondere bei drahtlosen Verbindungen ist die erreichbare Datenrate nicht nur vom Traffic abhängig.

Fairness

Fairness bedeutet, dass gleichzeitige Verbindungen in etwa die gleiche Datenrate bekommen, sich also die Leitung fair teilen.

- Idealerweise, der Router/Switch würde Pakete verschiedener Verbindungen round–robin senden. Das würde die berechnete Bandbreite entsprechend korrigieren, und wäre auch eine große Erleichterung beim TCP–Buffer–Bloat–Symptom, weil jede Verbindung ihren eigenen Puffer auffüllen würde.
- Der Vorschlag „Fair Queuing“ ist von 1985, und es gibt Router, die das auch implementieren (als konfigurierbare Option) — aber leider z.B. nicht für DSLAMs, die haben FIFO–Queuing
- Daraus ergibt sich die Notwendigkeit, auch für FIFO–Queuing einen stabilen Algorithmus zu finden
- Der Algorithmus muss sich geänderten Bedingungen schnell anpassen. Insbesondere bei drahtlosen Verbindungen ist die erreichbare Datenrate nicht nur vom Traffic abhängig.

Fairness

Fairness bedeutet, dass gleichzeitige Verbindungen in etwa die gleiche Datenrate bekommen, sich also die Leitung fair teilen.

- Idealerweise, der Router/Switch würde Pakete verschiedener Verbindungen round–robin senden. Das würde die berechnete Bandbreite entsprechend korrigieren, und wäre auch eine große Erleichterung beim TCP–Buffer–Bloat–Symptom, weil jede Verbindung ihren eigenen Puffer auffüllen würde.
- Der Vorschlag „Fair Queuing“ ist von 1985, und es gibt Router, die das auch implementieren (als konfigurierbare Option) — aber leider z.B. nicht für DSLAMs, die haben FIFO–Queuing
- Daraus ergibt sich die Notwendigkeit, auch für FIFO–Queuing einen stabilen Algorithmus zu finden.
- Der Algorithmus muss sich geänderten Bedingungen schnell anpassen. Insbesondere bei drahtlosen Verbindungen ist die erreichbare Datenrate nicht nur vom Traffic abhängig.

Fairness

Fairness bedeutet, dass gleichzeitige Verbindungen in etwa die gleiche Datenrate bekommen, sich also die Leitung fair teilen.

- Idealerweise, der Router/Switch würde Pakete verschiedener Verbindungen round–robin senden. Das würde die berechnete Bandbreite entsprechend korrigieren, und wäre auch eine große Erleichterung beim TCP–Buffer–Bloat–Symptom, weil jede Verbindung ihren eigenen Puffer auffüllen würde.
- Der Vorschlag „Fair Queuing“ ist von 1985, und es gibt Router, die das auch implementieren (als konfigurierbare Option) — aber leider z.B. nicht für DSLAMs, die haben FIFO–Queuing
- Daraus ergibt sich die Notwendigkeit, auch für FIFO–Queuing einen stabilen Algorithmus zu finden
- Der Algorithmus muss sich geänderten Bedingungen schnell anpassen. Insbesondere bei drahtlosen Verbindungen ist die erreichbare Datenrate nicht nur vom Traffic abhängig.

Fairness

Fairness bedeutet, dass gleichzeitige Verbindungen in etwa die gleiche Datenrate bekommen, sich also die Leitung fair teilen.

- Idealerweise, der Router/Switch würde Pakete verschiedener Verbindungen round–robin senden. Das würde die berechnete Bandbreite entsprechend korrigieren, und wäre auch eine große Erleichterung beim TCP–Buffer–Bloat–Symptom, weil jede Verbindung ihren eigenen Puffer auffüllen würde.
- Der Vorschlag „Fair Queuing“ ist von 1985, und es gibt Router, die das auch implementieren (als konfigurierbare Option) — aber leider z.B. nicht für DSLAMs, die haben FIFO–Queuing
- Daraus ergibt sich die Notwendigkeit, auch für FIFO–Queuing einen stabilen Algorithmus zu finden
- Der Algorithmus muss sich geänderten Bedingungen schnell anpassen. Insbesondere bei drahtlosen Verbindungen ist die erreichbare Datenrate nicht nur vom Traffic abhängig.

net2o–Flusskontrolle — Fair Queuing Router

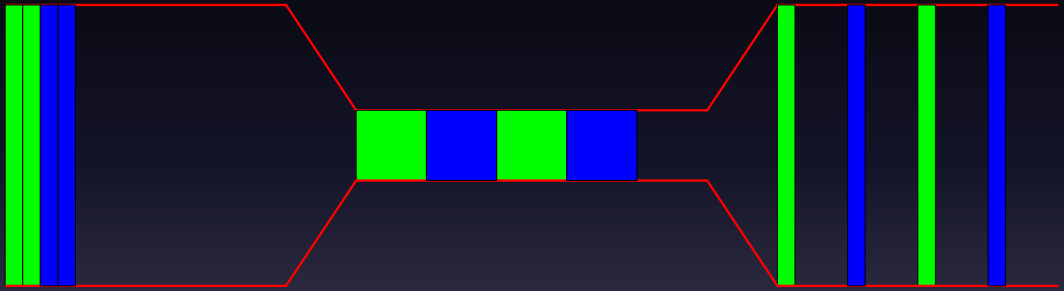


Abbildung : Fair queuing sorgt für korrete Messung der verfügbaren Bandbreite

net2o–Flusskontrolle — FIFO Router

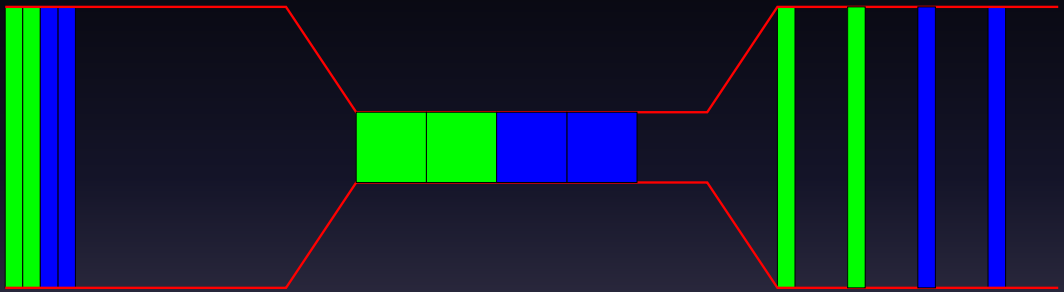


Abbildung : Unfaires FIFO queuing sorgt dafür, dass die gemessene Rate doppelt so hoch gemessen wird

Fairness I

- Um die Stabilität bei unfairem Queuing zu verbessern, muss ich den P–Regulator zu einem echten PID–Regulator ausbauen
- Der integrale Part ist der aufsummierte „Slack“ (im Puffer), den ich niedrig halten möchte, und der D–Anteil spiegelt die Veränderung dieses „Slacks“ von einer Messung zur nächsten wieder
- Ich verwende beide Komponenten, um die Rate beim Sender zu reduzieren, das sorgt für deutlich bessere Fairness
- Der I–Anteil ist exponentiell (maximal Faktor 16)

Fairness I

- Um die Stabilität bei unfairem Queuing zu verbessern, muss ich den P–Regulator zu einem echten PID–Regulator ausbauen
- Der integrale Part ist der aufsummierte „Slack“ (im Puffer), den ich niedrig halten möchte, und der D–Anteil spiegelt die Veränderung dieses „Slacks“ von einer Messung zur nächsten wieder
- Ich verwende beide Komponenten, um die Rate beim Sender zu reduzieren, das sorgt für deutlich bessere Fairness
- Der I–Anteil ist exponentiell (maximal Faktor 16)

Fairness I

- Um die Stabilität bei unfairem Queuing zu verbessern, muss ich den P–Regulator zu einem echten PID–Regulator ausbauen
- Der integrale Part ist der aufsummierte „Slack“ (im Puffer), den ich niedrig halten möchte, und der D–Anteil spiegelt die Veränderung dieses „Slacks“ von einer Messung zur nächsten wieder
- Ich verwende beide Komponenten, um die Rate beim Sender zu reduzieren, das sorgt für deutlich bessere Fairness
- Der I–Anteil ist exponentiell (maximal Faktor 16)

Fairness I

- Um die Stabilität bei unfairem Queuing zu verbessern, muss ich den P–Regulator zu einem echten PID–Regulator ausbauen
- Der integrale Part ist der aufsummierte „Slack“ (im Puffer), den ich niedrig halten möchte, und der D–Anteil spiegelt die Veränderung dieses „Slacks“ von einer Messung zur nächsten wieder
- Ich verwende beide Komponenten, um die Rate beim Sender zu reduzieren, das sorgt für deutlich bessere Fairness
- Der I–Anteil ist exponentiell (maximal Faktor 16)

Fairness I

- Um die Stabilität bei unfairem Queuing zu verbessern, muss ich den P–Regulator zu einem echten PID–Regulator ausbauen
- Der integrale Part ist der aufsummierte „Slack“ (im Puffer), den ich niedrig halten möchte, und der D–Anteil spiegelt die Veränderung dieses „Slacks“ von einer Messung zur nächsten wieder
- Ich verwende beide Komponenten, um die Rate beim Sender zu reduzieren, das sorgt für deutlich bessere Fairness
- Der I–Anteil ist exponentiell (maximal Faktor 16)

$$s_{exp} = 2^{\frac{slack}{T}} \quad \text{where } T = \max(10ms, \max(slacks))$$

Fairness D

- Um den differenziellen Anteil zu messen, messe ich, wie der Slack wächst vom ersten zum letzten Burst in einem Messzyklus (das sind 4 Bursts)
- Das wird dann hochgerechnet auf die Pakete, die insgesamt unterwegs sind (also von den gemessenen Paketen bis zu den gerade eben gesendeten Paketen)
- Der so errechnete Korrekturfaktor klingt dann langsam ab, mit einem RTD als Zeitkonstante, damit es nicht zu schnell hoch geht mit der Geschwindigkeit

  $\max(slacks)/10ms$ wird benutzt, um den Algorithmus mehr oder weniger aggressiv zu machen
- Das so ermittelte Δt wird sowohl zur Rate (auch ein Δt) für einen Burst addiert,

Fairness D

- Um den differenziellen Anteil zu messen, messe ich, wie der Slack wächst vom ersten zum letzten Burst in einem Messzyklus (das sind 4 Bursts)
- Das wird dann hochgerechnet auf die Pakete, die insgesamt unterwegs sind (also von den gemessenen Paketen bis zu den gerade eben gesendeten Paketen)
- Der so errechnete Korrekturfaktor klingt dann langsam ab, mit einem RTD als Zeitkonstante, damit es nicht zu schnell hoch geht mit der Geschwindigkeit
  $\max(slacks)/10ms$ wird benutzt, um den Algorithmus mehr oder weniger aggressiv zu machen
- Das so ermittelte Δt wird sowohl zur Rate (auch ein Δt) für einen Burst addiert,

Fairness D

- Um den differenziellen Anteil zu messen, messe ich, wie der Slack wächst vom ersten zum letzten Burst in einem Messzyklus (das sind 4 Bursts)
- Das wird dann hochgerechnet auf die Pakete, die insgesamt unterwegs sind (also von den gemessenen Paketen bis zu den gerade eben gesendeten Paketen)
- Der so errechnete Korrekturfaktor klingt dann langsam ab, mit einem RTD als Zeitkonstante, damit es nicht zu schnell hoch geht mit der Geschwindigkeit
- $\max(slacks)/10ms$ wird benutzt, um den Algorithmus mehr oder weniger aggressiv zu machen
- Das so ermittelte Δt wird sowohl zur Rate (auch ein Δt) für einen Burst addiert,

Fairness D

- Um den differenziellen Anteil zu messen, messe ich, wie der Slack wächst vom ersten zum letzten Burst in einem Messzyklus (das sind 4 Bursts)
- Das wird dann hochgerechnet auf die Pakete, die insgesamt unterwegs sind (also von den gemessenen Paketen bis zu den gerade eben gesendeten Paketen)
- Der so errechnete Korrekturfaktor klingt dann langsam ab, mit einem RTD als Zeitkonstante, damit es nicht zu schnell hoch geht mit der Geschwindigkeit
- $\max(slacks)/10ms$ wird benutzt, um den Algorithmus mehr oder weniger aggressiv zu machen
- Das so ermittelte Δt wird sowohl zur Rate (auch ein Δt) für einen Burst addiert,

Fairness D

- Um den differenziellen Anteil zu messen, messe ich, wie der Slack wächst vom ersten zum letzten Burst in einem Messzyklus (das sind 4 Bursts)
- Das wird dann hochgerechnet auf die Pakete, die insgesamt unterwegs sind (also von den gemessenen Paketen bis zu den gerade eben gesendeten Paketen)
- Der so errechnete Korrekturfaktor klingt dann langsam ab, mit einem RTD als Zeitkonstante, damit es nicht zu schnell hoch geht mit der Geschwindigkeit
- $\max(slacks)/10ms$ wird benutzt, um den Algorithmus mehr oder weniger aggressiv zu machen
- Das so ermittelte Δt wird sowohl zur Rate (auch ein Δt) für einen Burst addiert, als auch als zusätzliche Wartezeit bis zum nächsten Burst hinzugefügt

VDSL

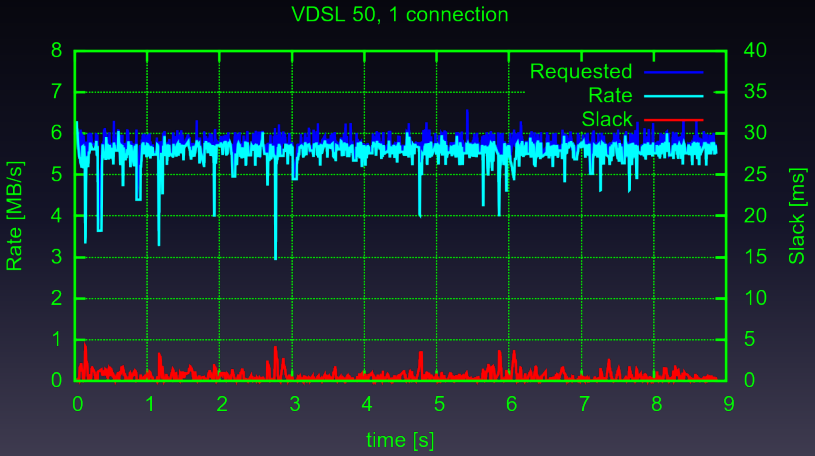


Abbildung : Eine Verbindung über VDSL–50

VDSL, Congestion

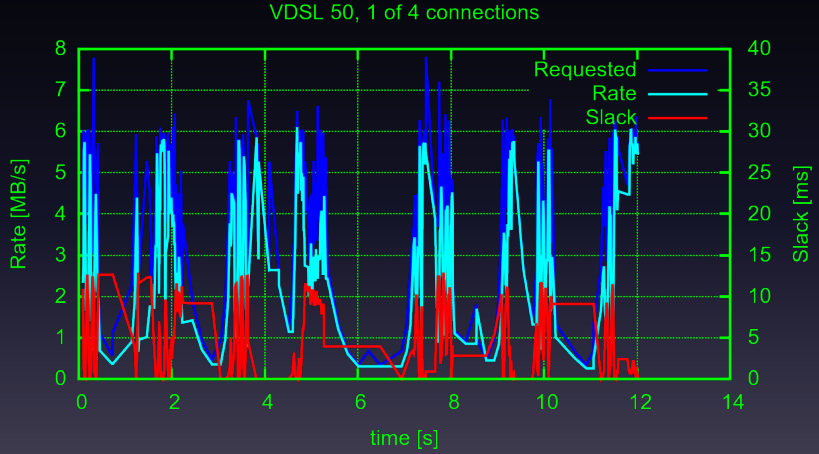


Abbildung : Eine von vier Verbindungen über VDSL–50

Unzuverlässiges Luftkabel (WLAN)

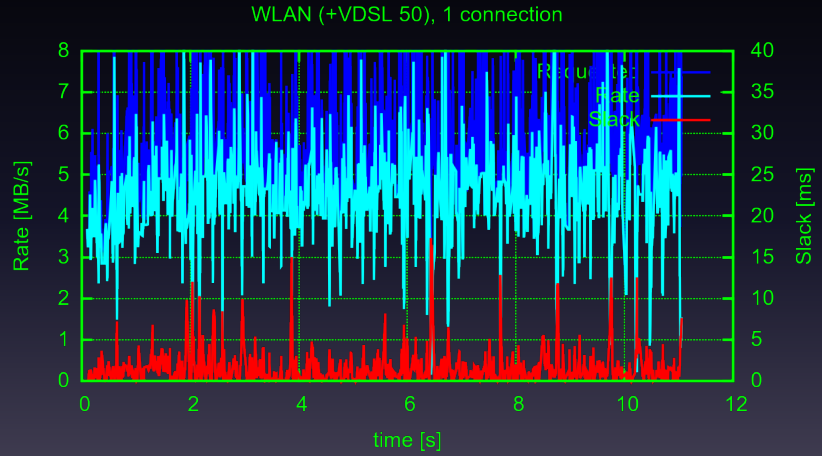


Abbildung : Eine Verbindung über WLAN

Unzuverlässiges Luftkabel, Congestion

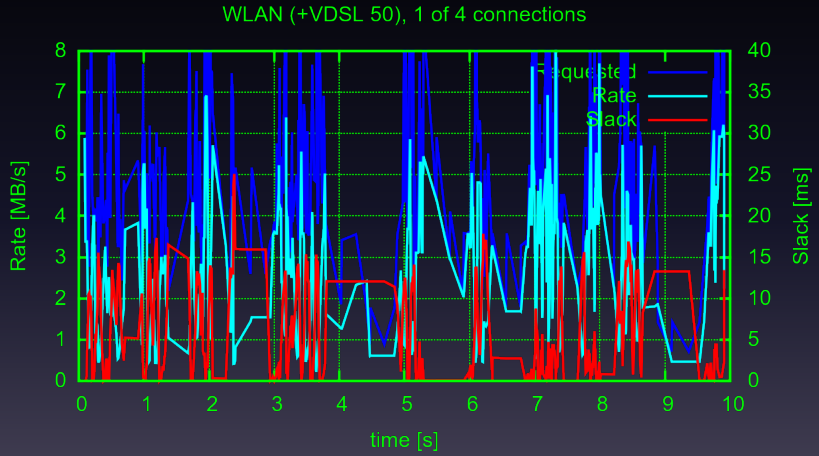


Abbildung : Eine von vier Verbindungen über WLAN

LAN, 1GBE

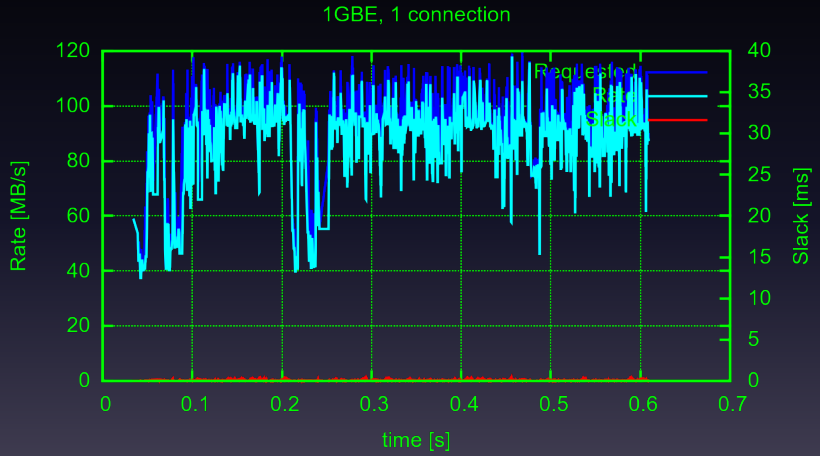


Abbildung : Eine Verbindung über 1GBE

LAN 1GBE, Congestion (4 servers)

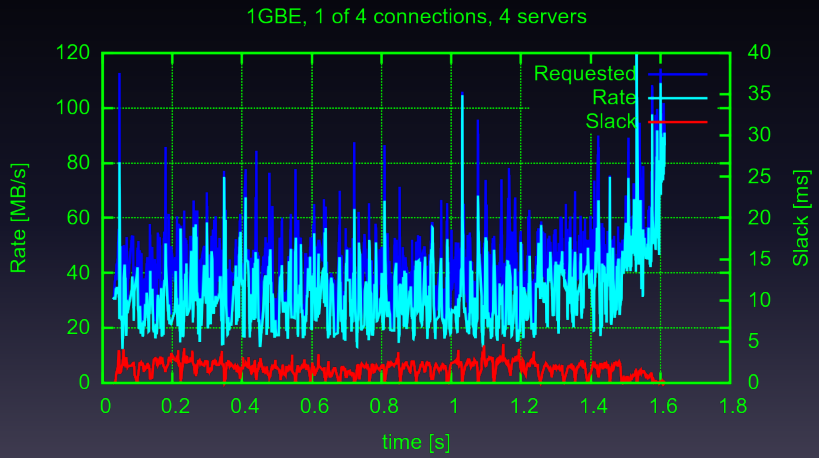


Abbildung : Eine Verbindung von vier über 1GBE

LAN 1GBE, Congestion (1 server)

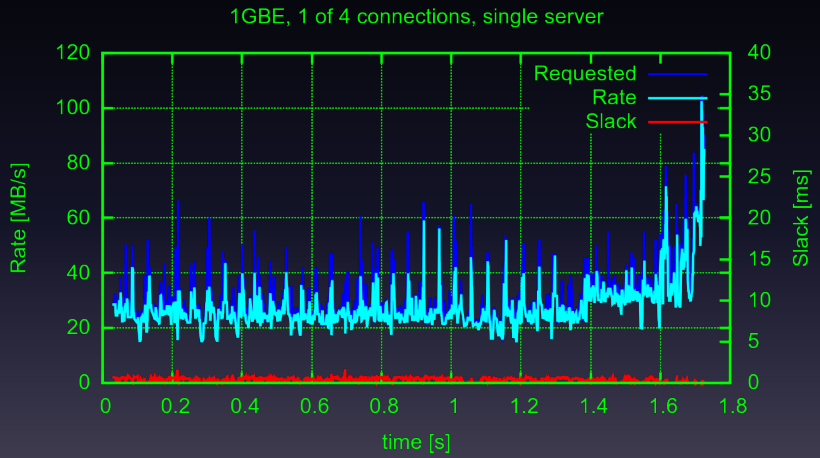


Abbildung : Eine von vier Verbindungen über 1GBE, fair queuing

Flusskontrolle — Resümee

- Die Flusskontrolle funktioniert, aber Fair Queuing (überall!) wäre ein echter Fortschritt
- Der primäre Algorithmus ist komplett anders als die ganze Konkurrenz: Ich messe die verfügbare Bandbreite
- Die Skalierbarkeit für ganz langsame Verbindungen ist schlecht: Bursts sind 8 Pakete lang
- Traffic bei Stau ohne fair queuing ist nicht voll zufriedenstellend

Flusskontrolle — Resümee

- Die Flusskontrolle funktioniert, aber Fair Queuing (überall!) wäre ein echter Fortschritt
- Der primäre Algorithmus ist komplett anders als die ganze Konkurrenz: Ich messe die verfügbare Bandbreite
- Die Skalierbarkeit für ganz langsame Verbindungen ist schlecht: Bursts sind 8 Pakete lang
- Traffic bei Stau ohne fair queuing ist nicht voll zufriedenstellend

Flusskontrolle — Resümee

- Die Flusskontrolle funktioniert, aber Fair Queuing (überall!) wäre ein echter Fortschritt
- Der primäre Algorithmus ist komplett anders als die ganze Konkurrenz: Ich messe die verfügbare Bandbreite
- Die Skalierbarkeit für ganz langsame Verbindungen ist schlecht: Bursts sind 8 Pakete lang
- Traffic bei Stau ohne fair queuing ist nicht voll zufriedenstellend

Flusskontrolle — Resümee

- Die Flusskontrolle funktioniert, aber Fair Queuing (überall!) wäre ein echter Fortschritt
- Der primäre Algorithmus ist komplett anders als die ganze Konkurrenz: Ich messe die verfügbare Bandbreite
- Die Skalierbarkeit für ganz langsame Verbindungen ist schlecht: Bursts sind 8 Pakete lang
- Traffic bei Stau ohne fair queuing ist nicht voll zufriedenstellend

Daten und Kommandos

- Daten von mehreren Dateien/Streams können in einer Verbindung gemultiplext werden
- Kommandos werden in Blöcken gesendet, also gibt es mehr als ein Kommando pro Block
- Die Kodierung verwendet die von Googles protobuf, also 7 Bits pro Byte; ist das MSB 1, kommt noch ein weiteres Byte
- Die Kommando–Maschine ist eine Stack–Maschine, also ein Forth–Subset

Daten und Kommandos

- Daten von mehreren Dateien/Streams können in einer Verbindung gemultiplext werden
- Kommandos werden in Blöcken gesendet, also gibt es mehr als ein Kommando pro Block
- Die Kodierung verwendet die von Googles protobuf, also 7 Bits pro Byte; ist das MSB 1, kommt noch ein weiteres Byte
- Die Kommando–Maschine ist eine Stack–Maschine, also ein Forth–Subset

Daten und Kommandos

- Daten von mehreren Dateien/Streams können in einer Verbindung gemultiplext werden
- Kommandos werden in Blöcken gesendet, also gibt es mehr als ein Kommando pro Block
- Die Kodierung verwendet die von Googles protobuf, also 7 Bits pro Byte; ist das MSB 1, kommt noch ein weiteres Byte
- Die Kommando–Maschine ist eine Stack–Maschine, also ein Forth–Subset

Daten und Kommandos

- Daten von mehreren Dateien/Streams können in einer Verbindung gemultiplext werden
- Kommandos werden in Blöcken gesendet, also gibt es mehr als ein Kommando pro Block
- Die Kodierung verwendet die von Googles protobuf, also 7 Bits pro Byte; ist das MSB 1, kommt noch ein weiteres Byte
- Die Kommando–Maschine ist eine Stack–Maschine, also ein Forth–Subset

Beispiel: Connection Request

```
"pk1" $, receive-tmpkey
nest[ timestamp1 lit, set-rtdelay gen-reply request-done ]nest $,
push-$ push' nest
tmpkey-request key-request
base lit, csize lit, dsize lit, map-request
```

Beispiel: Sauge drei Dateien

```
net2o-code
"Download test" $, type cr ( see-me )
get-ip $400 blocksize! $400 blockalign! stat( request-stats )
"net2o.fs" 0 lit, 0 lit, open-tracked-file
"data/2011-05-13_11-26-57-small.jpg" 0 lit, 1 lit, open-tracked-file
"data/2011-05-20_17-01-12-small.jpg" 0 lit, 2 lit, open-tracked-file
gen-total slurp-all-tracked-blocks send-chunks
0 lit, ok?
end-code
```

Beispiel: Antwort auf diesen Request

```
net2o-code
x" 360000000000000000000000000000019ED2" $, set-ip
 $E373 lit, 0 lit, track-size
$134299FF6F829E62 lit, $1A4 lit, 0 lit, set-stat
$9C65C lit, 1 lit, track-size
$130AFDAE900C649E lit, $1A4 lit, 1 lit, set-stat
$9D240 lit, 2 lit, track-size
$130AFDAE92CA4E25 lit, $1A4 lit, 2 lit, set-stat
$148000 lit, set-total
 $E373 lit, 0 lit, track-seek
$79000 lit, 1 lit, track-seek
$78C00 lit, 2 lit, track-seek
0 lit, ok
end-code
```

Verteilter Programmablauf über „Active Messages“

Die Kommandos sind so der Teil meines Vortrags, den die Nicht–Forther irgendwie gar nicht verstehen, deshalb da etwas mehr dazu:

- Das ist stark von Open Network Forth von Heinz Schnitter inspiriert. Nur tokenisiert.
- Die Antworten auf Anfragen sind ihrerseits wieder ausführbare Kommandos, die führt man einfach aus, und dann läuft das Programm weiter

  Das ist also asynchron und ohne irgendwelche Hooks und so: Man bastelt eine Message zusammen und schickt die weg
- Wenn die gewünschten Fragen nicht beantwortet werden, dann gibt es irgendwann

Verteilter Programmablauf über „Active Messages“

Die Kommandos sind so der Teil meines Vortrags, den die Nicht–Forther irgendwie gar nicht verstehen, deshalb da etwas mehr dazu:

- Das ist stark von Open Network Forth von HEINZ SCHNITTER inspiriert. Nur tokenisiert.
- Die Antworten auf Anfragen sind ihrerseits wieder ausführbare Kommandos, die führt man einfach aus, und dann läuft das Programm weiter

  Das ist also asynchron und ohne irgendwelche Hooks und so: Man bastelt eine Message zusammen und schickt die weg
- Wenn die gewünschten Fragen nicht beantwortet werden, dann gibt es irgendwann

Verteilter Programmablauf über „Active Messages“

Die Kommandos sind so der Teil meines Vortrags, den die Nicht–Forther irgendwie gar nicht verstehen, deshalb da etwas mehr dazu:

- Das ist stark von Open Network Forth von HEINZ SCHNITTER inspiriert. Nur tokenisiert.
- Die Antworten auf Anfragen sind ihrerseits wieder ausführbare Kommandos, die führt man einfach aus, und dann läuft das Programm weiter

  Das ist also asynchron und ohne irgendwelche Hooks und so: Man bastelt eine Message zusammen und schickt die weg
- Wenn die gewünschten Fragen nicht beantwortet werden, dann gibt es irgendwann

Verteilter Programmablauf über „Active Messages“

Die Kommandos sind so der Teil meines Vortrags, den die Nicht–Forther irgendwie gar nicht verstehen, deshalb da etwas mehr dazu:

- Das ist stark von Open Network Forth von HEINZ SCHNITTER inspiriert. Nur tokenisiert.
- Die Antworten auf Anfragen sind ihrerseits wieder ausführbare Kommandos, die führt man einfach aus, und dann läuft das Programm weiter
- Das ist also asynchron und ohne irgendwelche Hooks und so: Man bastelt eine Message zusammen und schickt die weg
- Wenn die gewünschten Fragen nicht beantwortet werden, dann gibt es irgendwann

Verteilter Programmablauf über „Active Messages“

Die Kommandos sind so der Teil meines Vortrags, den die Nicht–Forther irgendwie gar nicht verstehen, deshalb da etwas mehr dazu:

- Das ist stark von Open Network Forth von HEINZ SCHNITTER inspiriert. Nur tokenisiert.
- Die Antworten auf Anfragen sind ihrerseits wieder ausführbare Kommandos, die führt man einfach aus, und dann läuft das Programm weiter
- Das ist also asynchron und ohne irgendwelche Hooks und so: Man bastelt eine Message zusammen und schickt die weg
- Wenn die gewünschten Fragen nicht beantwortet werden, dann gibt es irgendwann einen Timeout

Verteilte Daten

- Jedes Datenobjekt ist eine „Datei“
- Datenobjekte werden über ihren Hash adressiert, und de zugehörigen Metadaten sind „Tags“
- Metadaten sind als verteilter Hash–Tree organisiert
- Die Daten im Hash–Tree werden durch Replikation verteilt
- Effiziente Distribution von Daten ist wichtig!

Verteilte Daten

- Jedes Datenobjekt ist eine „Datei“
- Datenobjekte werden über ihren Hash adressiert, und de zugehörigen Metadaten sind „Tags“
- Metadaten sind als verteilter Hash–Tree organisiert
- Die Daten im Hash–Tree werden durch Replikation verteilt
- Effiziente Distribution von Daten ist wichtig!

Verteilte Daten

- Jedes Datenobjekt ist eine „Datei“
- Datenobjekte werden über ihren Hash adressiert, und de zugehörigen Metadaten sind „Tags“
- Metadaten sind als verteilter Hash–Tree organisiert
- Die Daten im Hash–Tree werden durch Replikation verteilt
- Effiziente Distribution von Daten ist wichtig!

Verteilte Daten

- Jedes Datenobjekt ist eine „Datei“
- Datenobjekte werden über ihren Hash adressiert, und de zugehörigen Metadaten sind „Tags“
- Metadaten sind als verteilter Hash–Tree organisiert
- Die Daten im Hash–Tree werden durch Replikation verteilt
- Effiziente Distribution von Daten ist wichtig!

Verteilte Daten

- Jedes Datenobjekt ist eine „Datei“
- Datenobjekte werden über ihren Hash adressiert, und de zugehörigen Metadaten sind „Tags“
- Metadaten sind als verteilter Hash–Tree organisiert
- Die Daten im Hash–Tree werden durch Replikation verteilt
- Effiziente Distribution von Daten ist wichtig!

Effiziente Datenverteilung

Puzzle: Wie effizient kann man Daten (z.B. einen Video–Stream) in einem P2P–Netzwerk? Unter der Annahme, dass alle Peers gleich sind, und die Bandbreite haben, einen Stream in Echtzeit hochzuladen

- Offensichtliche Topologie: Die Eimerkette — damit sieht man, dass jeder Knoten die Daten durchleitet — eine 1:1–Relation dessen, was man bekommt und man sendet
- Eimerkette: $O(n)$ Latenz, $O\left(\frac{1}{n}\right)$ Robustheit (jeder Knoten bricht die Kette)
- Vorschlag: Baumstruktur statt Kette, z.B. ein Quad–Tree. Die Wurzel teilt die Daten in vier Teile, jeder wird in einem Zweig des Baums verteilt. Die Blätter verteilen die Daten an die anderen drei Zweige.
- Im Quad–Tree–Fall hat jeder Knoten nur 3 Nachbarn, 4 Quellen und 4 Senken

Effiziente Datenverteilung

Puzzle: Wie effizient kann man Daten (z.B. einen Video–Stream) in einem P2P–Netzwerk? Unter der Annahme, dass alle Peers gleich sind, und die Bandbreite haben, einen Stream in Echtzeit hochzuladen

- Offensichtliche Topologie: Die Eimerkette — damit sieht man, dass jeder Knoten die Daten durchleitet — eine 1:1–Relation dessen, was man bekommt und man sendet
  - Eimerkette: $O(n)$ Latenz, $O\left(\frac{1}{n}\right)$ Robustheit (jeder Knoten bricht die Kette)
  - Vorschlag: Baumstruktur statt Kette, z.B. ein Quad–Tree. Die Wurzel teilt die Daten in vier Teile, jeder wird in einem Zweig des Baums verteilt. Die Blätter verteilen die Daten an die anderen drei Zweige.
- Im Quad–Tree–Fall hat jeder Knoten nur 3 Nachbarn, 4 Quellen und 4 Senken

Effiziente Datenverteilung

Puzzle: Wie effizient kann man Daten (z.B. einen Video–Stream) in einem P2P–Netzwerk? Unter der Annahme, dass alle Peers gleich sind, und die Bandbreite haben, einen Stream in Echtzeit hochzuladen

- Offensichtliche Topologie: Die Eimerkette — damit sieht man, dass jeder Knoten die Daten durchleitet — eine 1:1–Relation dessen, was man bekommt und man sendet
- Eimerkette: $O(n)$ Latenz, $O\left(\frac{1}{n}\right)$ Robustheit (jeder Knoten bricht die Kette)
- Vorschlag: Baumstruktur statt Kette, z.B. ein Quad–Tree. Die Wurzel teilt die Daten in vier Teile, jeder wird in einem Zweig des Baums verteilt. Die Blätter verteilen die Daten an die anderen drei Zweige.
- Im Quad–Tree–Fall hat jeder Knoten nur 3 Nachbarn, 4 Quellen und 4 Senken

Effiziente Datenverteilung

Puzzle: Wie effizient kann man Daten (z.B. einen Video–Stream) in einem P2P–Netzwerk? Unter der Annahme, dass alle Peers gleich sind, und die Bandbreite haben, einen Stream in Echtzeit hochzuladen

- Offensichtliche Topologie: Die Eimerkette — damit sieht man, dass jeder Knoten die Daten durchleitet — eine 1:1–Relation dessen, was man bekommt und man sendet
- Eimerkette: $O(n)$ Latenz, $O\left(\frac{1}{n}\right)$ Robustheit (jeder Knoten bricht die Kette)
- Vorschlag: Baumstruktur statt Kette, z.B. ein Quad–Tree. Die Wurzel teilt die Daten in vier Teile, jeder wird in einem Zweig des Baums verteilt. Die Blätter verteilen die Daten an die anderen drei Zweite.
- Im Quad–Tree Fall hat jeder Knoten nur 3 Nachbarn: 1 Quelle und 2 Senken

Effiziente Datenverteilung

Puzzle: Wie effizient kann man Daten (z.B. einen Video–Stream) in einem P2P–Netzwerk? Unter der Annahme, dass alle Peers gleich sind, und die Bandbreite haben, einen Stream in Echtzeit hochzuladen

- Offensichtliche Topologie: Die Eimerkette — damit sieht man, dass jeder Knoten die Daten durchleitet — eine 1:1–Relation dessen, was man bekommt und man sendet
- Eimerkette: $O(n)$ Latenz, $O\left(\frac{1}{n}\right)$ Robustheit (jeder Knoten bricht die Kette)
- Vorschlag: Baumstruktur statt Kette, z.B. ein Quad–Tree. Die Wurzel teilt die Daten in vier Teile, jeder wird in einem Zweig des Baums verteilt. Die Blätter verteilen die Daten an die anderen drei Zweite.
- Im Quad–Tree–Fall hat jeder Knoten nur 8 Nachbarn: 4 Quellen und 4 Senken

Baum als Verteilernetzwerk

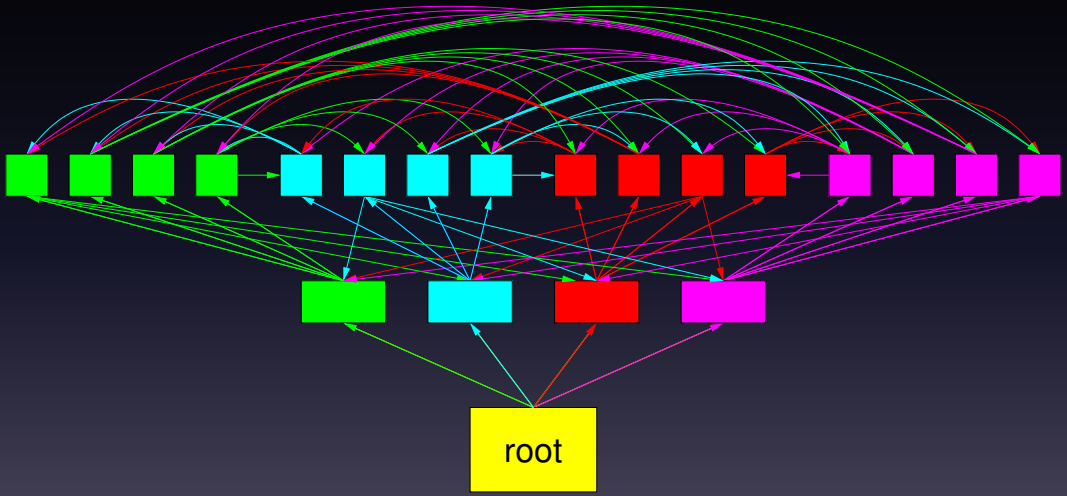


Abbildung : Lawinen–Verteilung mit einem quad–tree der Tiefe 2

Mögliche Performance

- Ein Baum mit einer größeren Basis reduziert die Latenz. Beispiel: Um einen Justin–Bieber–Tweet an die 50 Millionen Followers zu verteilen, braucht ein Binärbaum 25,5 Hops im Durchschnitt, ein Quad–Tree 12,8 Hops und ein Oct–Tree 8,5 Hops.
- Eine typische hop–zu–hop–Zeit in einem Land (z.B. in Deutschland) ist knapp 20ms. Internationale Hops können bis zu 250ms lang sein. Unter der Annahme, dass nur ein internationaler Hop drin ist, dauert es typischerweise nur 500ms, um Justin Bibers Gebabble in einem Quad–Tree zu verteilen.
- Daumenregel: *bandwidth* = *latency*, wenn es also 20ms braucht, zum nächsten Hop zu kommen, dann repliziert jeder Knoten für 20ms Daten — wenn wir den Baum breiter machen, dominiert der lineare Aufwand der Replikation, wenn wir den Baum schmaler machen, die Hop–zu–Hop–Zeit dominiert.
- Der baumartige Graph reduziert die Anzahl der bekannten Knoten

Mögliche Performance

- Ein Baum mit einer größeren Basis reduziert die Latenz. Beispiel: Um einen Justin–Bieber–Tweet an die 50 Millionen Followers zu verteilen, braucht ein Binärbaum 25,5 Hops im Durchschnitt, ein Quad–Tree 12,8 Hops und ein Oct–Tree 8,5 Hops.
- Eine typische hop–zu–hop–Zeit in einem Land (z.B. in Deutschland) ist knapp 20ms. Internationale Hops können bis zu 250ms lang sein. Unter der Annahme, dass nur ein internationaler Hop drin ist, dauert es typischerweise nur 500ms, um Justin Bibers Gebabble in einem Quad–Tree zu verteilen.
- Daumenregel: *bandwidth* = *latency*, wenn es also 20ms braucht, zum nächsten Hop zu kommen, dann repliziert jeder Knoten für 20ms Daten — wenn wir den Baum breiter machen, dominiert der lineare Aufwand der Replikation, wenn wir den Baum schmaler machen, die Hop–zu–Hop–Zeit dominiert.
- Der baumartige Graph reduziert die Anzahl der bekannten Knoten

Mögliche Performance

- Ein Baum mit einer größeren Basis reduziert die Latenz. Beispiel: Um einen Justin–Bieber–Tweet an die 50 Millionen Followers zu verteilen, braucht ein Binärbaum 25,5 Hops im Durchschnitt, ein Quad–Tree 12,8 Hops und ein Oct–Tree 8,5 Hops.
- Eine typische hop–zu–hop–Zeit in einem Land (z.B. in Deutschland) ist knapp 20ms. Internationale Hops können bis zu 250ms lang sein. Unter der Annahme, dass nur ein internationaler Hop drin ist, dauert es typischerweise nur 500ms, um Justin Bibers Gebabble in einem Quad–Tree zu verteilen.
- Daumenregel: *bandwidth* = *latency*, wenn es also 20ms braucht, zum nächsten Hop zu kommen, dann repliziert jeder Knoten für 20ms Daten — wenn wir den Baum breiter machen, dominiert der lineare Aufwand der Replikation, wenn wir den Baum schmaler machen, die Hop–zu–Hop–Zeit dominiert
- Der baumartige Graph reduziert die Anzahl der bekannten Knoten

Mögliche Performance

- Ein Baum mit einer größeren Basis reduziert die Latenz. Beispiel: Um einen Justin–Bieber–Tweet an die 50 Millionen Followers zu verteilen, braucht ein Binärbaum 25,5 Hops im Durchschnitt, ein Quad–Tree 12,8 Hops und ein Oct–Tree 8,5 Hops.
- Eine typische hop–zu–hop–Zeit in einem Land (z.B. in Deutschland) ist knapp 20ms. Internationale Hops können bis zu 250ms lang sein. Unter der Annahme, dass nur ein internationaler Hop drin ist, dauert es typischerweise nur 500ms, um Justin Bibers Gebabble in einem Quad–Tree zu verteilen.
- Daumenregel: $\mathit{bandwidth} = \mathit{latency}$, wenn es also 20ms braucht, zum nächsten Hop zu kommen, dann repliziert jeder Knoten für 20ms Daten — wenn wir den Baum breiter machen, dominiert der lineare Aufwand der Replikation, wenn wir den Baum schmaler machen, die Hop–zu–Hop–Zeit dominiert
- Der baumartige Graph reduziert die Anzahl der bekannten Knoten

Metadatenschutz

- Ein weltweit verteilter Hash–Tree ist von jemanden mit genügend Geld 100% beobachtbar
- Private Bäume teilen nur Daten innerhalb einer Gruppe Leute (die alle den Schlüssel kennen): „Dunkle Bäume in eine dunklen Wald“
- Nutze verschiedene Identitäten für unterschiedliche Gruppen, jede nur der jeweiligen Gruppe bekannt: „Dunkler sozialer Graph“
- Die öffentliche ID ist nur für erste Kontakte und Dinge, die man veröffentlichen will (PR)
- Nutze Nachbarschaftsverhältnisse um das Verteilen von Daten zu begrenzen — z.B. ein Knoten, der beim Datenverteilen teilnimmt, kennt nur die Nachbarn in diesem Baum

Metadatenschutz

- Ein weltweit verteilter Hash–Tree ist von jemanden mit genügend Geld 100% beobachtbar
- Private Bäume teilen nur Daten innerhalb einer Gruppe Leute (die alle den Schlüssel kennen): „Dunkle Bäume in eine dunklen Wald“
- Nutze verschiedene Identitäten für unterschiedliche Gruppen, jede nur der jeweiligen Gruppe bekannt: „Dunkler sozialer Graph“
- Die öffentliche ID ist nur für erste Kontakte und Dinge, die man veröffentlichen will (PR)
- Nutze Nachbarschaftsverhältnisse um das Verteilen von Daten zu begrenzen — z. B. ein Knoten, der beim Datenverteilen teilnimmt, kennt nur die Nachbarn in diesem Baum

Metadatenschutz

- Ein weltweit verteilter Hash–Tree ist von jemanden mit genügend Geld 100% beobachtbar
- Private Bäume teilen nur Daten innerhalb einer Gruppe Leute (die alle den Schlüssel kennen): „Dunkle Bäume in eine dunklen Wald“
- Nutze verschiedene Identitäten für unterschiedliche Gruppen, jede nur der jeweiligen Gruppe bekannt: „Dunkler sozialer Graph“
- Die öffentliche ID ist nur für erste Kontakte und Dinge, die man veröffentlichen will (PR)
- Nutze Nachbarschaftsverhältnisse um das Verteilen von Daten zu begrenzen — z.B. ein Knoten, der beim Datenverteilen teilnimmt, kennt nur die Nachbarn in diesem Baum

Metadatenschutz

- Ein weltweit verteilter Hash–Tree ist von jemanden mit genügend Geld 100% beobachtbar
- Private Bäume teilen nur Daten innerhalb einer Gruppe Leute (die alle den Schlüssel kennen): „Dunkle Bäume in eine dunklen Wald“
- Nutze verschiedene Identitäten für unterschiedliche Gruppen, jede nur der jeweiligen Gruppe bekannt: „Dunkler sozialer Graph“
- Die öffentliche ID ist nur für erste Kontakte und Dinge, die man veröffentlichen will (PR)
- Nutze Nachbarschaftsverhältnisse um das Verteilen von Daten zu begrenzen — z. B. ein Knoten, der beim Datenverteilen zufällig nur die Nachbarn in diesem Baum

Metadatenschutz

- Ein weltweit verteilter Hash–Tree ist von jemanden mit genügend Geld 100% beobachtbar
- Private Bäume teilen nur Daten innerhalb einer Gruppe Leute (die alle den Schlüssel kennen): „Dunkle Bäume in eine dunklen Wald“
- Nutze verschiedene Identitäten für unterschiedliche Gruppen, jede nur der jeweiligen Gruppe bekannt: „Dunkler sozialer Graph“
- Die öffentliche ID ist nur für erste Kontakte und Dinge, die man veröffentlichen will (PR)
- Nutze Nachbarschaftsverhältnisse um das Verteilen von Daten zu begrenzen — z.B. ein Knoten, der beim Datenverteilen teilnimmt, kennt nur die Nachbaren in diesem Baum

For Further Reading I

Bernd Paysan

net2o source repository

http://fossil.net2o.de/net2o

Shay Gueron, Vlad Krasnov

The fragility of AES-GCM authentication algorithm

http://eprint.iacr.org/2013/157.pdf

Markku-Juhani O. Saarinen

GCM, GHASH and Weak Keys

http://www.ecrypt.eu.org/hash2011/proceedings/hash2011_03.pdf