net 2.0 Netzwerkschichten

Reinventing the Internet

Bernd Paysan

Forth–Tagung 2016, Augsburg

Inhalt

net2o zusammengefasst

net2o besteht aus den folgenden 6 Konzepten (so ungefähr ähnlich wie Schichten, aber nicht wirklich):

2. Packete der Größe 2^n, Pfad–Switching, am Ziel in shared memory buffers
3. Ephemeral Schlüsselaustausch und Signaturen mit Ed25519, symmetrisch authentisierte Verschlüsselung, Hash und PRNG mit Keccak,
   symmetrische Blockverschlüsseung mit Threefish
4. Zeitgetriebene verzögerungsminimierte Flusskontrolle
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metdaten (distributed hash table)
7. Apps in einer Sandbox, um Inhalte anzuzeigen (to be done)

net2o zusammengefasst

net2o besteht aus den folgenden 6 Konzepten (so ungefähr ähnlich wie Schichten, aber nicht wirklich):

2. Packete der Größe 2^n, Pfad–Switching, am Ziel in shared memory buffers
3. Ephemeral Schlüsselaustausch und Signaturen mit Ed25519, symmetrisch authentisierte Verschlüsselung, Hash und PRNG mit Keccak,
   symmetrische Blockverschlüsseung mit Threefish
4. Zeitgetriebene verzögerungsminimierte Flusskontrolle
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metdaten (distributed hash table)
7. Apps in einer Sandbox, um Inhalte anzuzeigen (to be done)

net2o zusammengefasst

net2o besteht aus den folgenden 6 Konzepten (so ungefähr ähnlich wie Schichten, aber nicht wirklich):

2. Packete der Größe 2^n, Pfad–Switching, am Ziel in shared memory buffers
3. Ephemeral Schlüsselaustausch und Signaturen mit Ed25519, symmetrisch authentisierte Verschlüsselung, Hash und PRNG mit Keccak,
   symmetrische Blockverschlüsseung mit Threefish
4. Zeitgetriebene verzögerungsminimierte Flusskontrolle
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metdaten (distributed hash table)
7. Apps in einer Sandbox, um Inhalte anzuzeigen (to be done)

net2o zusammengefasst

net2o besteht aus den folgenden 6 Konzepten (so ungefähr ähnlich wie Schichten, aber nicht wirklich):

2. Packete der Größe 2^n, Pfad–Switching, am Ziel in shared memory buffers
3. Ephemeral Schlüsselaustausch und Signaturen mit Ed25519, symmetrisch authentisierte Verschlüsselung, Hash und PRNG mit Keccak,
   symmetrische Blockverschlüsseung mit Threefish
4. Zeitgetriebene verzögerungsminimierte Flusskontrolle
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metdaten (distributed hash table)
7. Apps in einer Sandbox, um Inhalte anzuzeigen (to be done)

net2o zusammengefasst

net2o besteht aus den folgenden 6 Konzepten (so ungefähr ähnlich wie Schichten, aber nicht wirklich):

2. Packete der Größe 2^n, Pfad–Switching, am Ziel in shared memory buffers
3. Ephemeral Schlüsselaustausch und Signaturen mit Ed25519, symmetrisch authentisierte Verschlüsselung, Hash und PRNG mit Keccak,
   symmetrische Blockverschlüsseung mit Threefish
4. Zeitgetriebene verzögerungsminimierte Flusskontrolle
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metdaten (distributed hash table)
7. Apps in einer Sandbox, um Inhalte anzuzeigen (to be done)

net2o zusammengefasst

net2o besteht aus den folgenden 6 Konzepten (so ungefähr ähnlich wie Schichten, aber nicht wirklich):

2. Packete der Größe 2^n, Pfad–Switching, am Ziel in shared memory buffers
3. Ephemeral Schlüsselaustausch und Signaturen mit Ed25519, symmetrisch authentisierte Verschlüsselung, Hash und PRNG mit Keccak,
   symmetrische Blockverschlüsseung mit Threefish
4. Zeitgetriebene verzögerungsminimierte Flusskontrolle
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metdaten (distributed hash table)
7. Apps in einer Sandbox, um Inhalte anzuzeigen (to be done)

net2o zusammengefasst

net2o besteht aus den folgenden 6 Konzepten (so ungefähr ähnlich wie Schichten, aber nicht wirklich):

2. Packete der Größe 2^n, Pfad–Switching, am Ziel in shared memory buffers
3. Ephemeral Schlüsselaustausch und Signaturen mit Ed25519, symmetrisch authentisierte Verschlüsselung, Hash und PRNG mit Keccak,
   symmetrische Blockverschlüsseung mit Threefish
4. Zeitgetriebene verzögerungsminimierte Flusskontrolle
5. Stack–orientierte tokenisierte Kommandos (RPC)
6. Verteilte Daten (Dateien) und Metdaten (distributed hash table)
7. Apps in einer Sandbox, um Inhalte anzuzeigen (to be done)

Ziele

Die Designziele von net2o sind

- leichtgewichtig, skalierbar und schnell
- einfach zu implementieren
- sicher
- medien/echtzeitfähig
- als Overlay über aktuellen Protokollen (UDP), aber auch als Ersatz des ganzen Stacks nutzbar

Ziele

Die Designziele von net2o sind

- leichtgewichtig, skalierbar und schnell
- einfach zu implementieren
- sicher
- medien/echtzeitfähig
- als Overlay über aktuellen Protokollen (UDP), aber auch als Ersatz des ganzen Stacks nutzbar

Ziele

Die Designziele von net2o sind

- leichtgewichtig, skalierbar und schnell
- einfach zu implementieren
- sicher
- medien/echtzeitfähig
- als Overlay über aktuellen Protokollen (UDP), aber auch als Ersatz des ganzen Stacks nutzbar

Ziele

Die Designziele von net2o sind

- leichtgewichtig, skalierbar und schnell
- einfach zu implementieren
- sicher
- medien/echtzeitfähig
- als Overlay über aktuellen Protokollen (UDP), aber auch als Ersatz des ganzen Stacks nutzbar

Ziele

Die Designziele von net2o sind

- leichtgewichtig, skalierbar und schnell
- einfach zu implementieren
- sicher
- medien/echtzeitfähig
- als Overlay über aktuellen Protokollen (UDP), aber auch als Ersatz des ganzen Stacks nutzbar

Ziele

Die Designziele von net2o sind

- leichtgewichtig, skalierbar und schnell
- einfach zu implementieren
- sicher
- medien/echtzeitfähig
- als Overlay über aktuellen Protokollen (UDP), aber auch als Ersatz des ganzen Stacks nutzbar

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

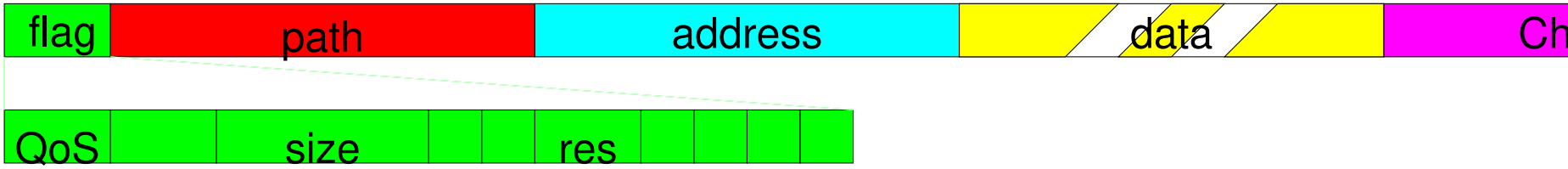
Path Switching

- 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 4
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
- Erschwerte Bedingungen im Netz 1.0: NAT und Firewalls... da muss man dann NAT Traversal machen (oder Firewall traversal bei IPv6)

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
- Erschwerte Bedingungen im Netz 1.0: NAT und Firewalls... da muss man dann NAT Traversal machen (oder Firewall traversal bei IPv6)

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
- Erschwerte Bedingungen im Netz 1.0: NAT und Firewalls... da muss man dann NAT Traversal machen (oder Firewall traversal bei IPv6)

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
- Erschwerte Bedingungen im Netz 1.0: NAT und Firewalls... da muss man dann NAT Traversal machen (oder Firewall traversal bei IPv6)

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
- Erschwerte Bedingungen im Netz 1.0: NAT und Firewalls... da muss man dann NAT Traversal machen (oder Firewall traversal bei IPv6)

Warum Source Routing

Drei mögliche Netztopologien

1. Leitungsvermittelt (POTS, virtuell: ATM, MPLS)
2. Eindeutige ID (IP)
3. Source Routing

- Trennung von Netzwerkequipment und Computer: schnelles, billiges, zustandsloses Equipment für Switching
- Die hierarchische Topologie ist gewissermaßen ein „Naturgesetz“: Menschen gruppieren sich in Clustern, und verbinden diese
- Ein Angriff auf das Netz ist bandbreitenbasiert, und kann relativ einfach („fair queuing“) abgewiesen werden
- Jedes Netzwerksegment kann seine Routing–Tags als Implementierungsdetails selbst interpretieren

Warum Source Routing

Drei mögliche Netztopologien

1. Leitungsvermittelt (POTS, virtuell: ATM, MPLS)
2. Eindeutige ID (IP)
3. Source Routing

- Trennung von Netzwerkequipment und Computer: schnelles, billiges, zustandsloses Equipment für Switching
- Die hierarchische Topologie ist gewissermaßen ein „Naturgesetz“: Menschen gruppieren sich in Clustern, und verbinden diese
- Ein Angriff auf das Netz ist bandbreitenbasiert, und kann relativ einfach („fair queuing“) abgewiesen werden
- Jedes Netzwerksegment kann seine Routing–Tags als Implementierungsdetails selbst interpretieren

Warum Source Routing

Drei mögliche Netztopologien

1. Leitungsvermittelt (POTS, virtuell: ATM, MPLS)
2. Eindeutige ID (IP)
3. Source Routing

- Trennung von Netzwerkequipment und Computer: schnelles, billiges, zustandsloses Equipment für Switching
- Die hierarchische Topologie ist gewissermaßen ein „Naturgesetz“: Menschen gruppieren sich in Clustern, und verbinden diese
- Ein Angriff auf das Netz ist bandbreitenbasiert, und kann relativ einfach („fair queuing“) abgewiesen werden
- Jedes Netzwerksegment kann seine Routing–Tags als Implementierungsdetails selbst interpretieren

Warum Source Routing

Drei mögliche Netztopologien

1. Leitungsvermittelt (POTS, virtuell: ATM, MPLS)
2. Eindeutige ID (IP)
3. Source Routing

- Trennung von Netzwerkequipment und Computer: schnelles, billiges, zustandsloses Equipment für Switching
- Die hierarchische Topologie ist gewissermaßen ein „Naturgesetz“: Menschen gruppieren sich in Clustern, und verbinden diese
- Ein Angriff auf das Netz ist bandbreitenbasiert, und kann relativ einfach („fair queuing“) abgewiesen werden
- Jedes Netzwerksegment kann seine Routing–Tags als Implementierungsdetails selbst interpretieren

Warum Source Routing

Drei mögliche Netztopologien

1. Leitungsvermittelt (POTS, virtuell: ATM, MPLS)
2. Eindeutige ID (IP)
3. Source Routing

- Trennung von Netzwerkequipment und Computer: schnelles, billiges, zustandsloses Equipment für Switching
- Die hierarchische Topologie ist gewissermaßen ein „Naturgesetz“: Menschen gruppieren sich in Clustern, und verbinden diese
- Ein Angriff auf das Netz ist bandbreitenbasiert, und kann relativ einfach („fair queuing“) abgewiesen werden
- Jedes Netzwerksegment kann seine Routing–Tags als Implementierungsdetails selbst interpretieren

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.

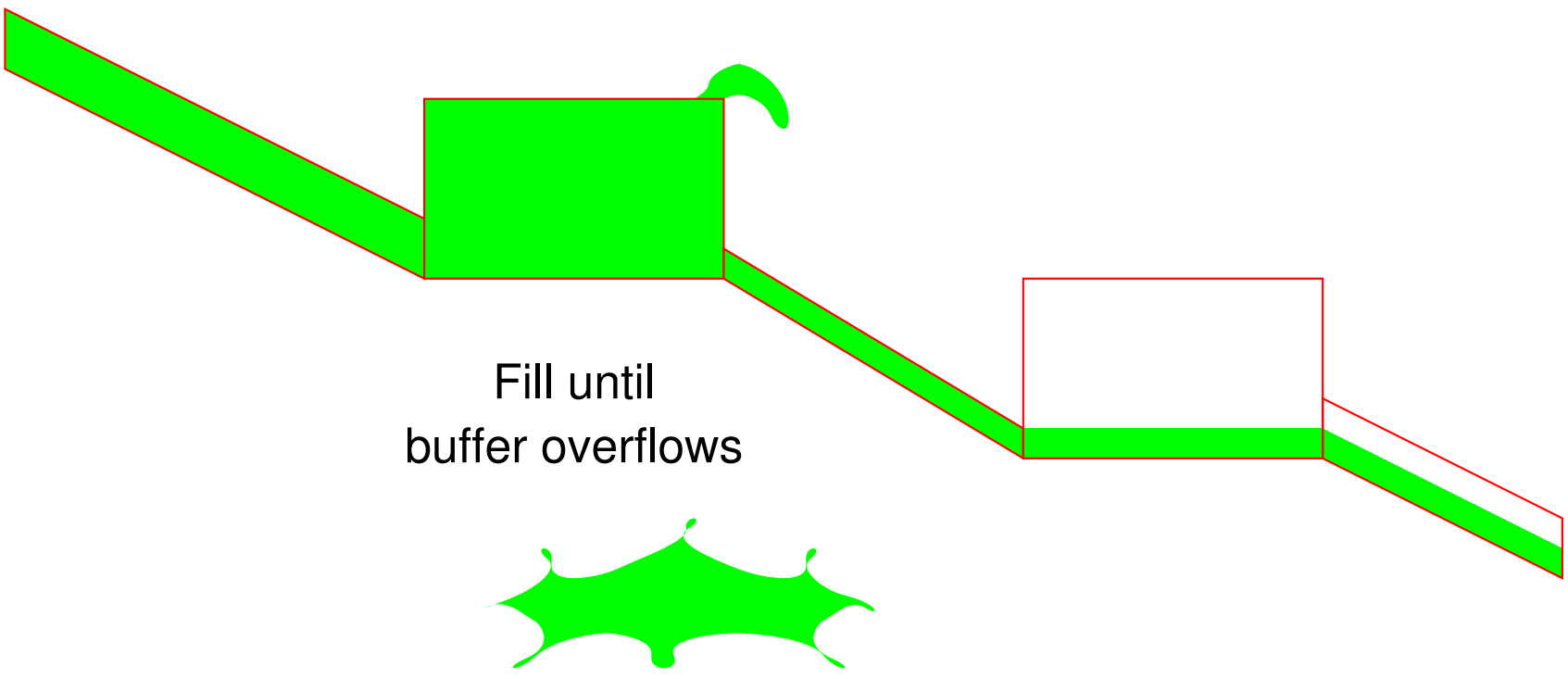


Abbildung: Buffer Bloat

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

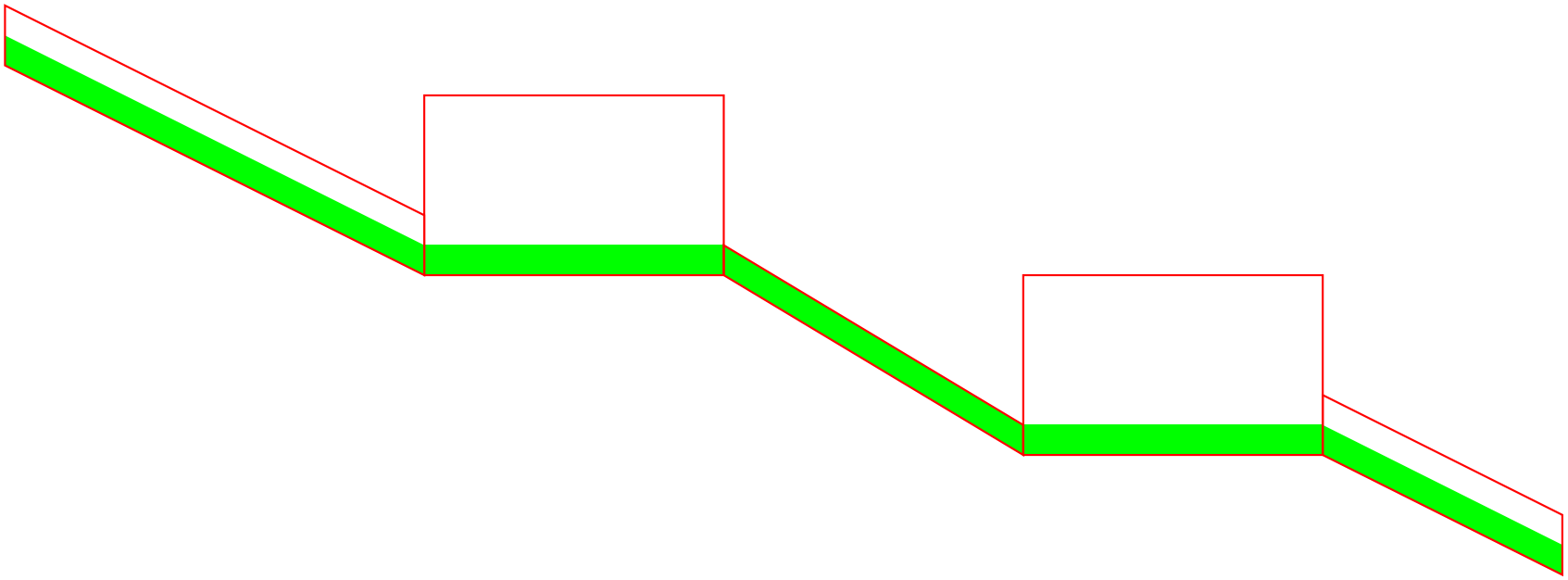


Abbildung: So soll es sein bei optimaler Flusskontrolle

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

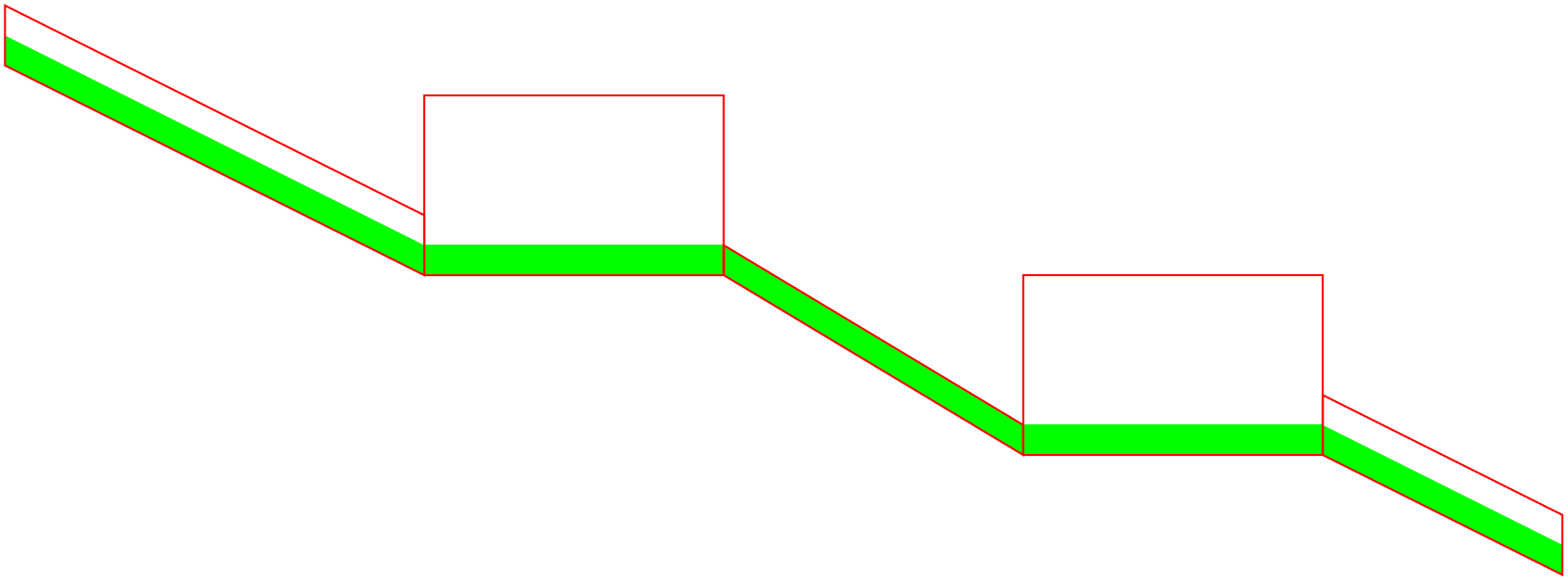


Abbildung: So soll es sein bei optimaler Flusskontrolle

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

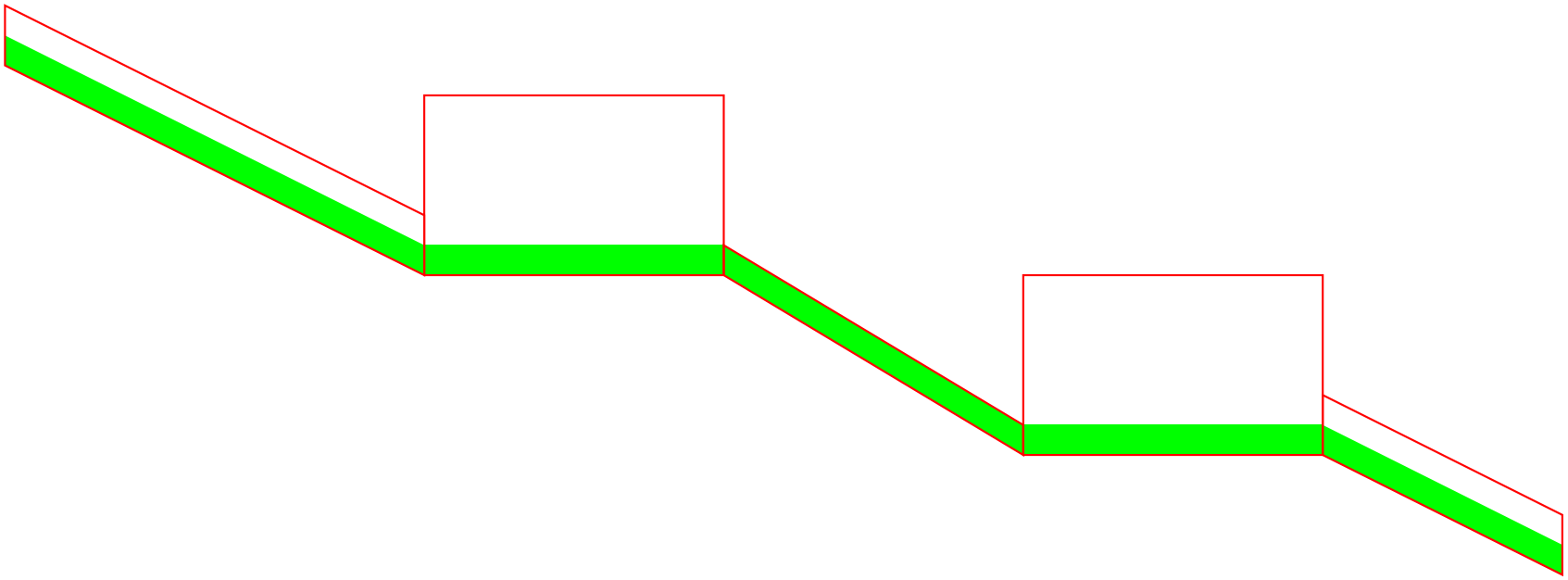


Abbildung: So soll es sein bei optimaler Flusskontrolle

net2o–Flusskontrolle

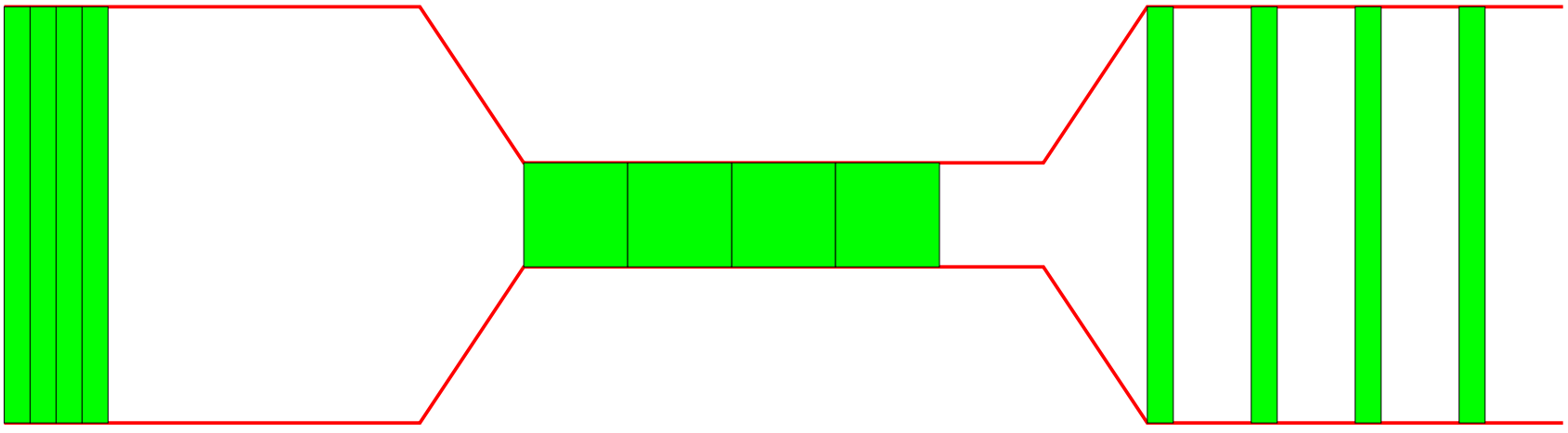


Abbildung: Miss den Flaschenhals mit einem Burst

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

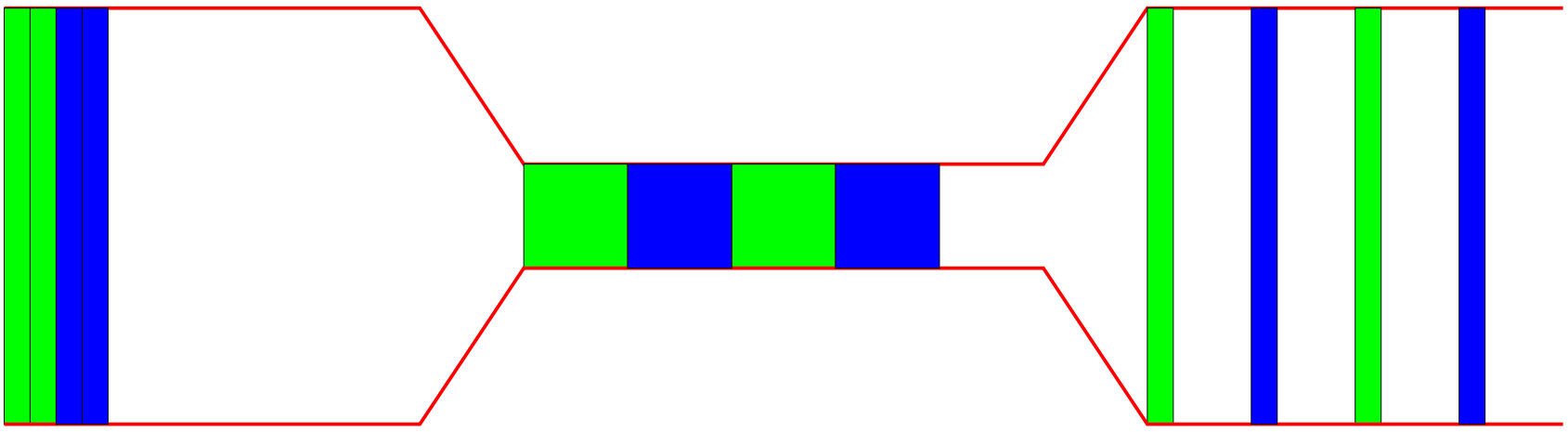


Abbildung: Fair queuing sorgt für korrete Messung der verfügbaren Bandbreite

net2o–Flusskontrolle — FIFO Router

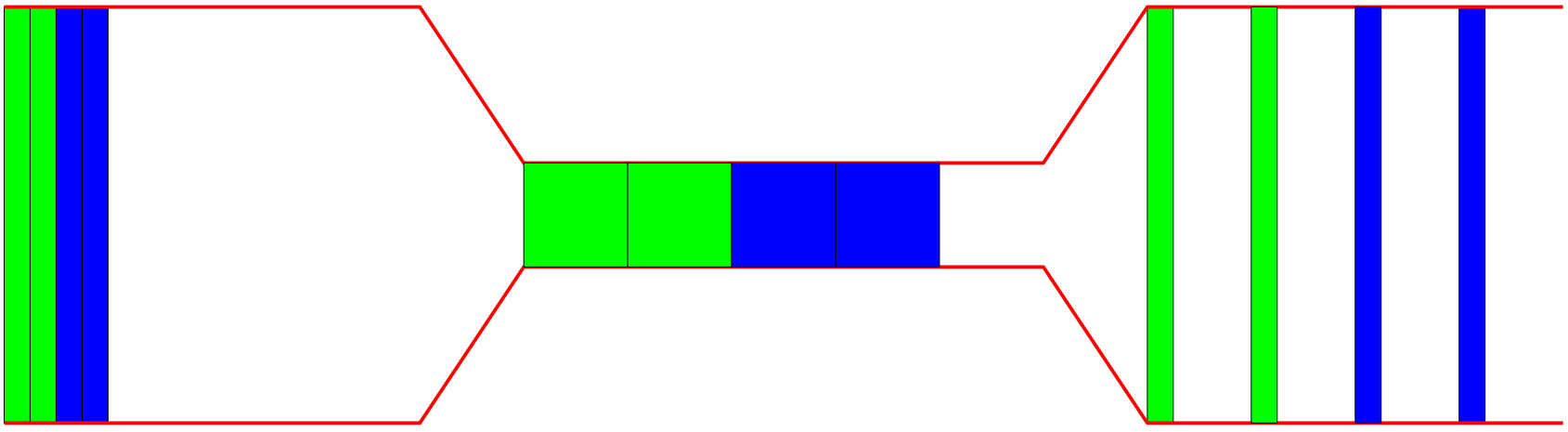


Abbildung: Unfaires FIFO queuing sorgt dafür, dass die gemessene Rate doppelt so hoch gemessen wird

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)

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

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 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 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 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, als auch als zusätzliche Wartezeit bis zum nächsten Burst hinzugefügt

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

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

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

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

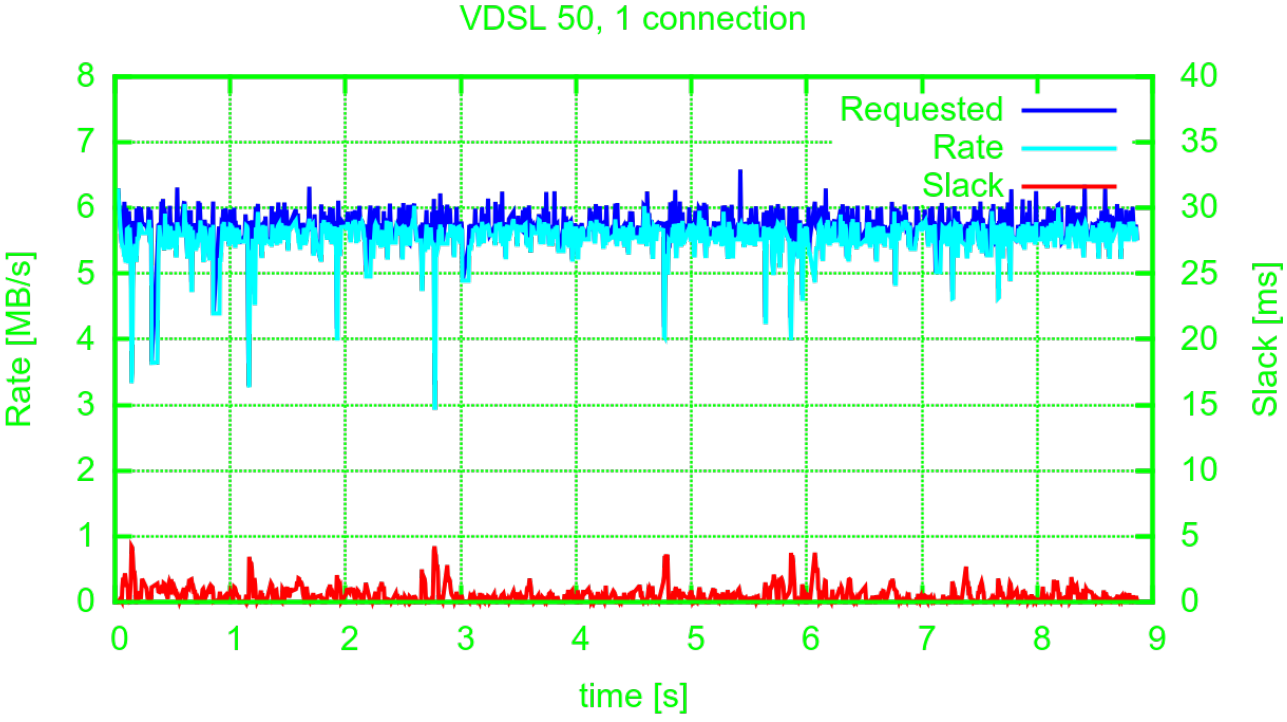


Abbildung: Eine Verbindung über VDSL–50

VDSL, Congestion

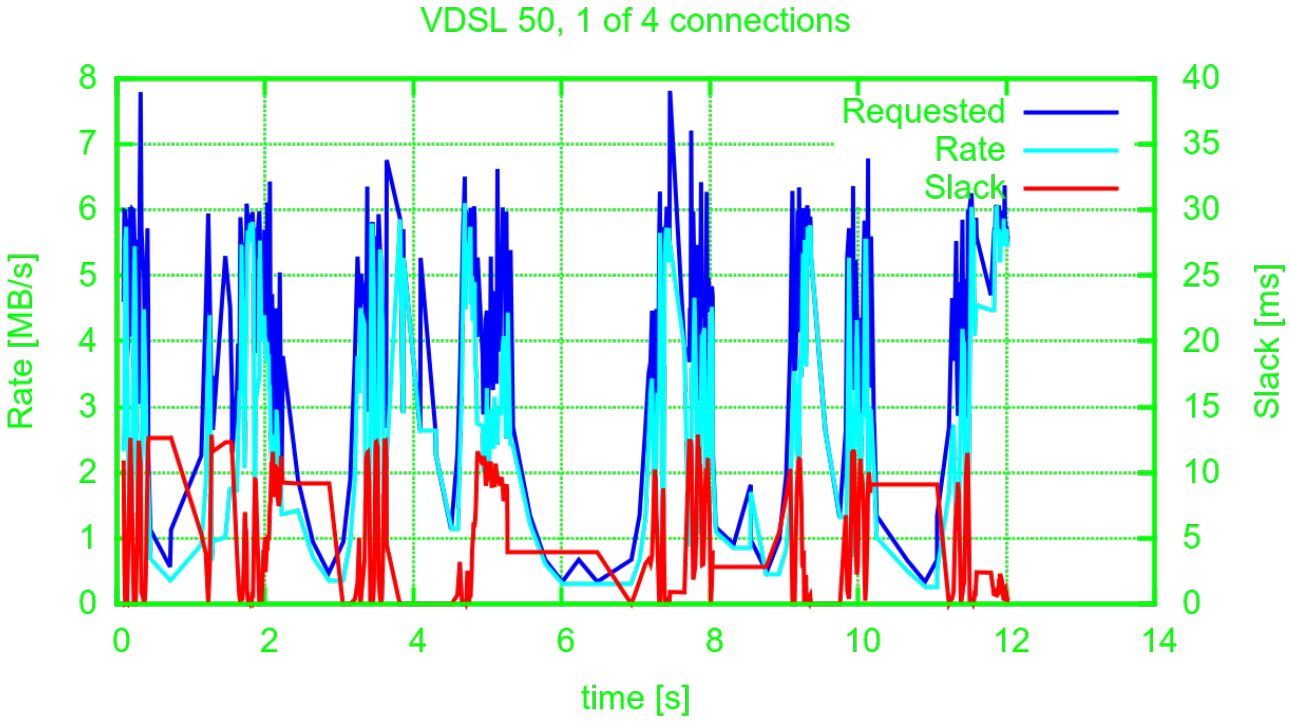


Abbildung: Eine von vier Verbindungen über VDSL–50

Unzuverlässiges Luftkabel (WLAN)

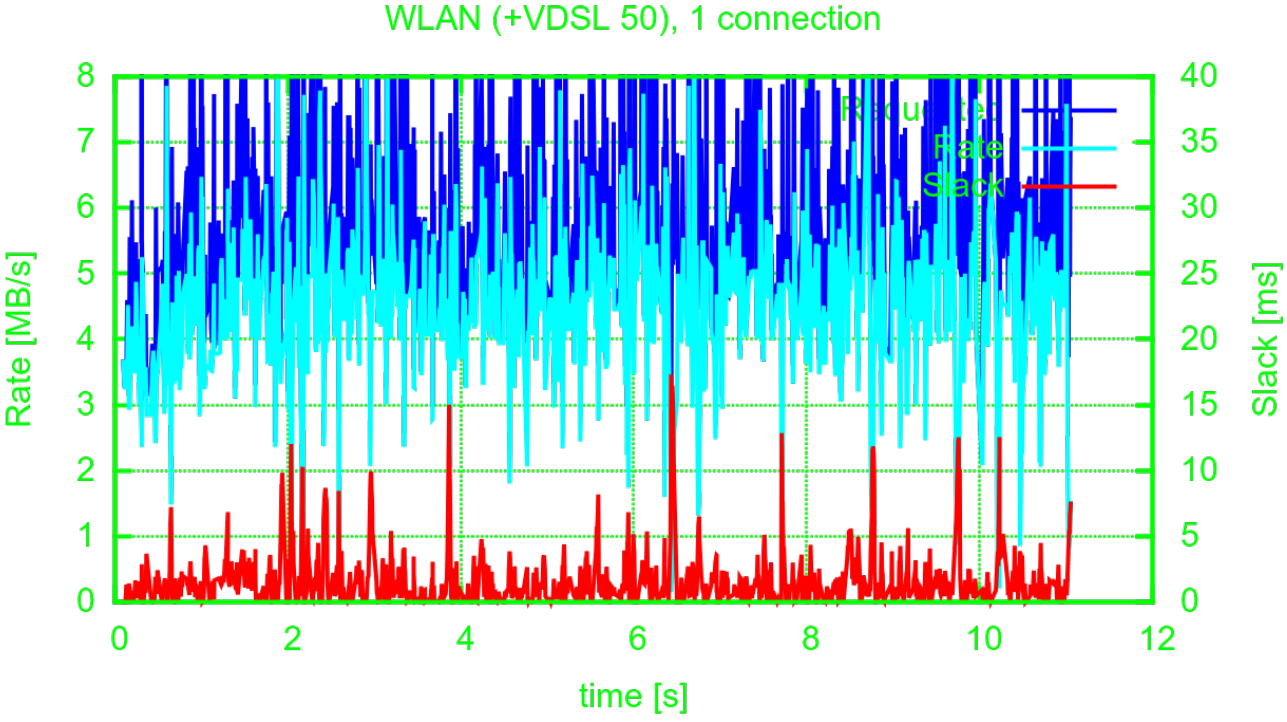


Abbildung: Eine Verbindung über WLAN

Unzuverlässiges Luftkabel, Congestion

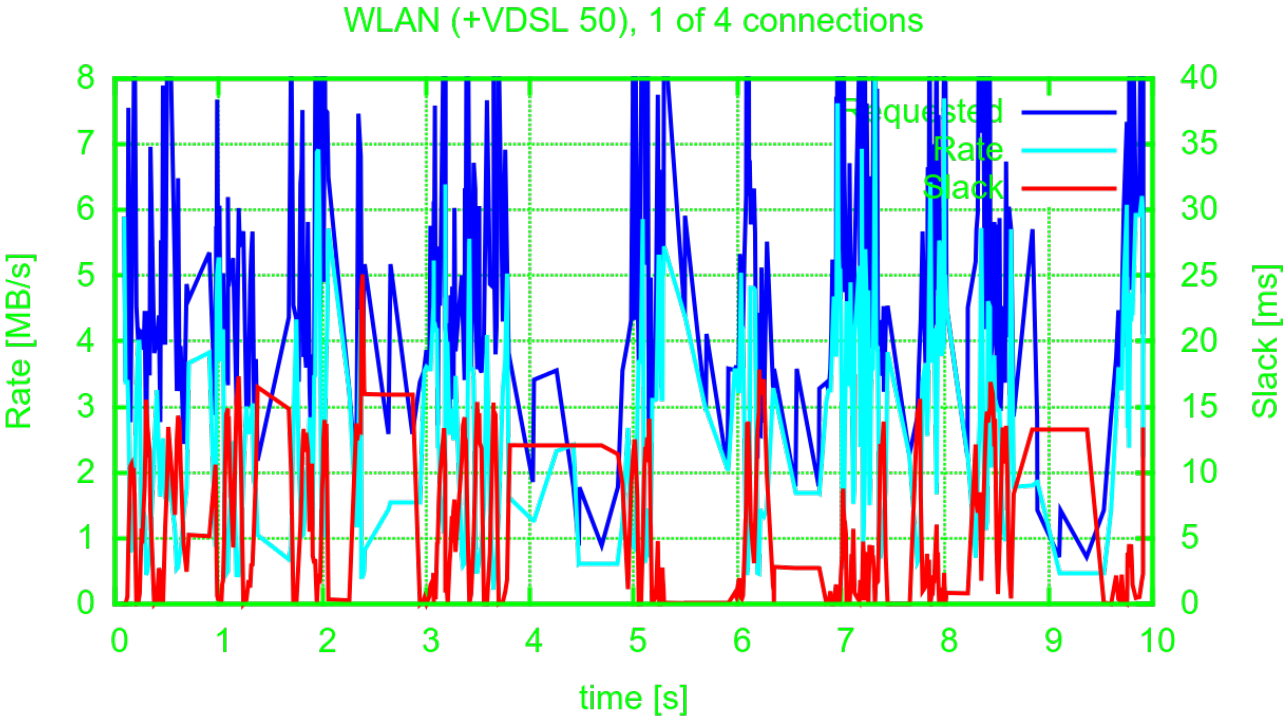


Abbildung: Eine von vier Verbindungen über WLAN

LAN, 1GBE

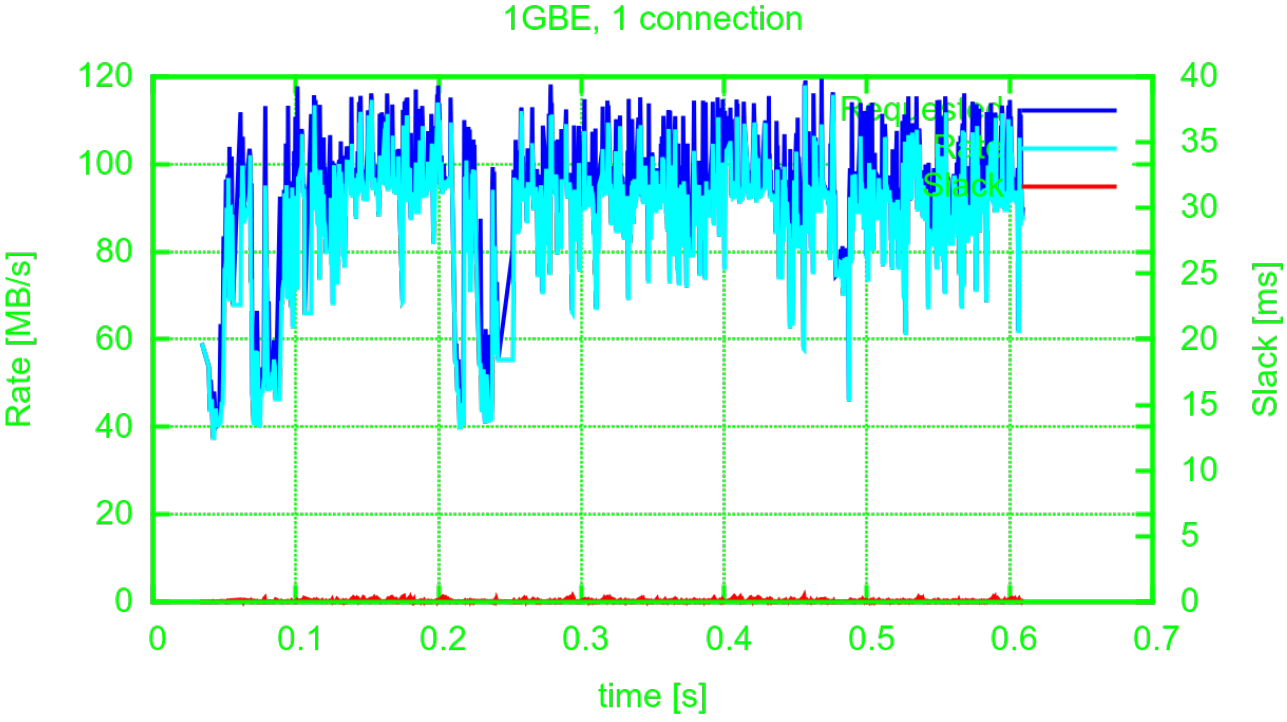


Abbildung: Eine Verbindung über 1GBE

LAN 1GBE, Congestion (4 servers)

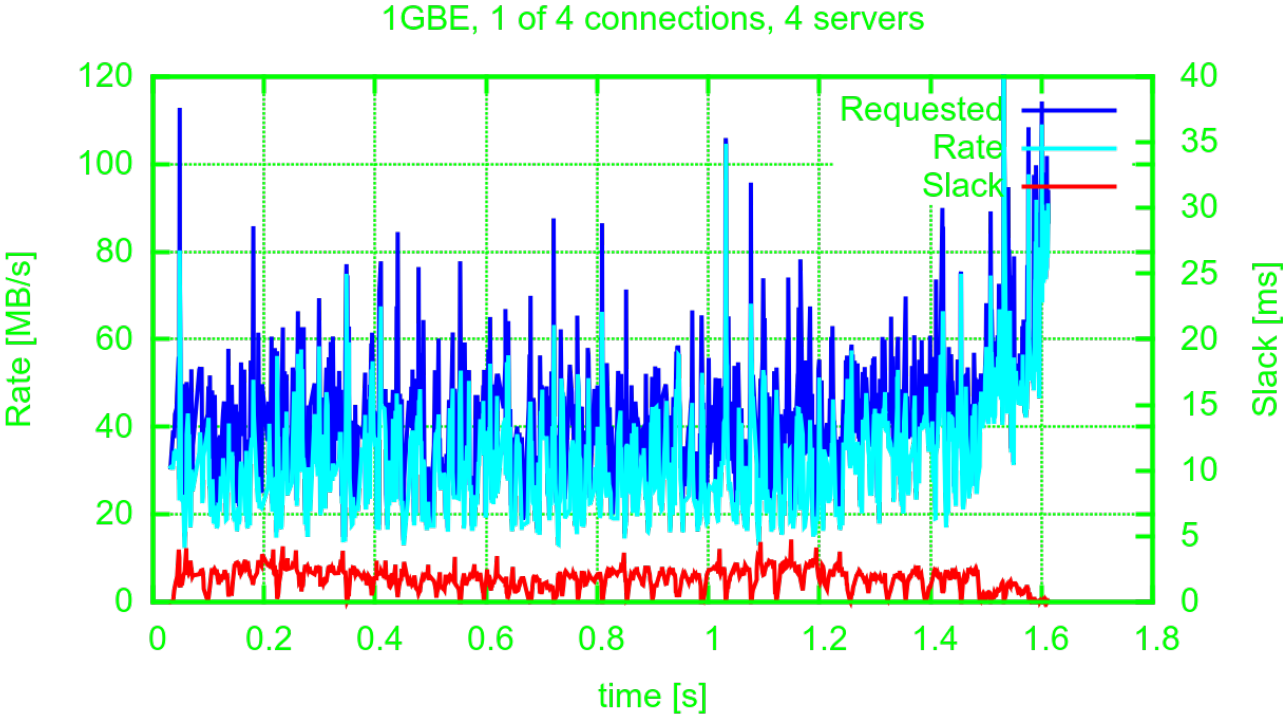


Abbildung: Eine Verbindung von vier über 1GBE

LAN 1GBE, Congestion (1 server)

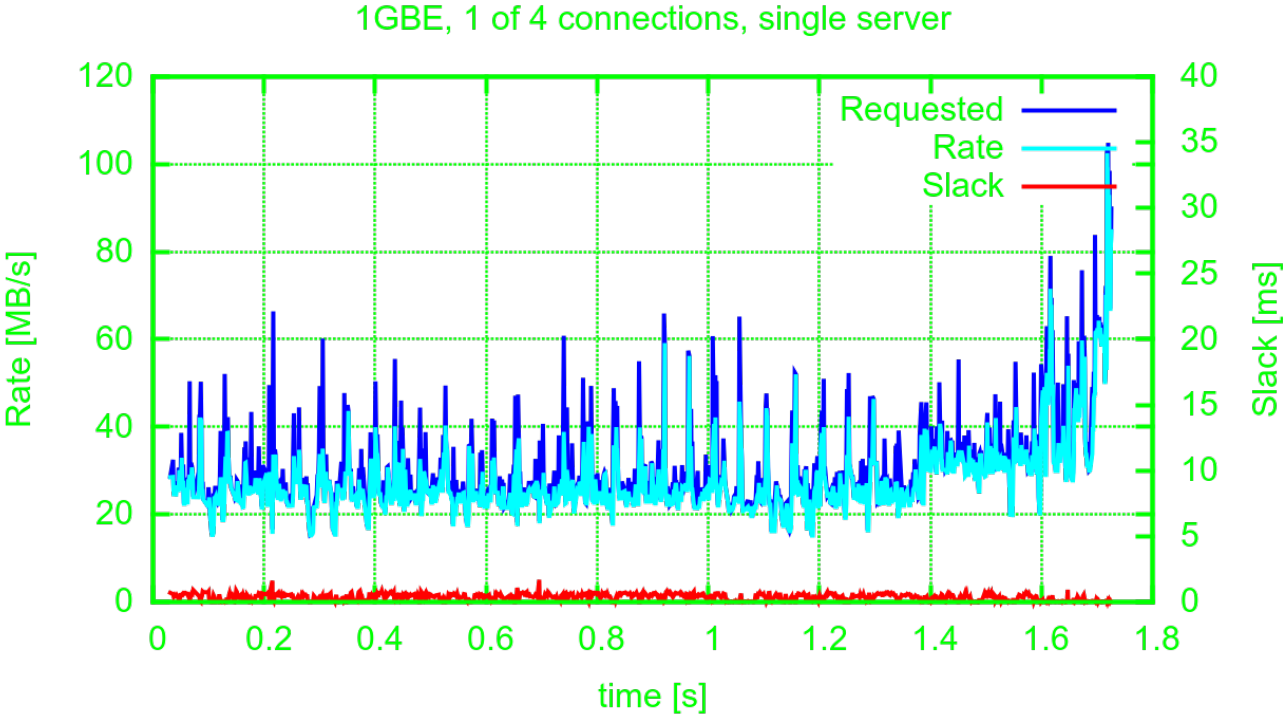


Abbildung: Eine von vier Verbindungen über 1GBE, fair queuing

Kommunikation im Forth–Stil

Anforderungen für sichere Kommunikation (sicher wie in „keine Exploits durch Fehlinterpretation“)

- Extrem einfacher Interpreter
- Erweiterbar, aber der Empfänger muss die Erweiterungen erlauben
- Universell, also nur ein Interpreter, den man auditen und verifizieren muss
- Die triviale Einfachheit macht es schwierig, das zu erklären

Kommunikation im Forth–Stil

Anforderungen für sichere Kommunikation (sicher wie in „keine Exploits durch Fehlinterpretation“)

- Extrem einfacher Interpreter
- Erweiterbar, aber der Empfänger muss die Erweiterungen erlauben
- Universell, also nur ein Interpreter, den man auditen und verifizieren muss
- Die triviale Einfachheit macht es schwierig, das zu erklären

Kommunikation im Forth–Stil

Anforderungen für sichere Kommunikation (sicher wie in „keine Exploits durch Fehlinterpretation“)

- Extrem einfacher Interpreter
- Erweiterbar, aber der Empfänger muss die Erweiterungen erlauben
- Universell, also nur ein Interpreter, den man auditen und verifizieren muss
- Die triviale Einfachheit macht es schwierig, das zu erklären

Kommunikation im Forth–Stil

Anforderungen für sichere Kommunikation (sicher wie in „keine Exploits durch Fehlinterpretation“)

- Extrem einfacher Interpreter
- Erweiterbar, aber der Empfänger muss die Erweiterungen erlauben
- Universell, also nur ein Interpreter, den man auditen und verifizieren muss
- Die triviale Einfachheit macht es schwierig, das zu erklären

Kommunikation im Forth–Stil

Anforderungen für sichere Kommunikation (sicher wie in „keine Exploits durch Fehlinterpretation“)

- Extrem einfacher Interpreter
- Erweiterbar, aber der Empfänger muss die Erweiterungen erlauben
- Universell, also nur ein Interpreter, den man auditen und verifizieren muss
- Die triviale Einfachheit macht es schwierig, das zu erklären

Grundlagen

- Fünf Datentypen: Integer (64 Bit mit und ohne Vorzeichen), Flag, String (generisches Byte–Array), IEEE float, Objekte.
- Instruktionen und Daten werden wie Protobuf[4] codiert (7 bits pro Byte, MSB=1 bedeutet „Daten gehen weiter“, höchstwertiger Teil zuerst)
- Vier Stacks: integer, float, objects, strings
- `endwith` und `endcmd` um Objekt–Message–Blöcke und Kommandos zu beenden
- `oswap` um das aktuelle Objekt für's Einfügen in das Vater–Objekt auf den Objekt–Stack zu legen
- `words` für Reflections (die Wörter werden mit Token–Nummer, Identifier und Stack–Effekt gelistet, damit automatische Bindings erzeugbar sind)

Grundlagen

- Fünf Datentypen: Integer (64 Bit mit und ohne Vorzeichen), Flag, String (generisches Byte–Array), IEEE float, Objekte.
- Instruktionen und Daten werden wie Protobuf[4] codiert (7 bits pro Byte, MSB=1 bedeutet „Daten gehen weiter“, höchstwertiger Teil zuerst)
- Vier Stacks: integer, float, objects, strings
- `endwith` und `endcmd` um Objekt–Message–Blöcke und Kommandos zu beenden
- `oswap` um das aktuelle Objekt für's Einfügen in das Vater–Objekt auf den Objekt–Stack zu legen
- `words` für Reflections (die Wörter werden mit Token–Nummer, Identifier und Stack–Effekt gelistet, damit automatische Bindings erzeugbar sind)

Grundlagen

- Fünf Datentypen: Integer (64 Bit mit und ohne Vorzeichen), Flag, String (generisches Byte–Array), IEEE float, Objekte.
- Instruktionen und Daten werden wie Protobuf[4] codiert (7 bits pro Byte, MSB=1 bedeutet „Daten gehen weiter“, höchstwertiger Teil zuerst)
- Vier Stacks: integer, float, objects, strings
- `endwith` und `endcmd` um Objekt–Message–Blöcke und Kommandos zu beenden
- `oswap` um das aktuelle Objekt für's Einfügen in das Vater–Objekt auf den Objekt–Stack zu legen
- `words` für Reflections (die Wörter werden mit Token–Nummer, Identifier und Stack–Effekt gelistet, damit automatische Bindings erzeugbar sind)

Grundlagen

- Fünf Datentypen: Integer (64 Bit mit und ohne Vorzeichen), Flag, String (generisches Byte–Array), IEEE float, Objekte.
- Instruktionen und Daten werden wie Protobuf[4] codiert (7 bits pro Byte, MSB=1 bedeutet „Daten gehen weiter“, höchstwertiger Teil zuerst)
- Vier Stacks: integer, float, objects, strings
- `endwith` und `endcmd` um Objekt–Message–Blöcke und Kommandos zu beenden
- `oswap` um das aktuelle Objekt für's Einfügen in das Vater–Objekt auf den Objekt–Stack zu legen
- `words` für Reflections (die Wörter werden mit Token–Nummer, Identifier und Stack–Effekt gelistet, damit automatische Bindings erzeugbar sind)

Grundlagen

- Fünf Datentypen: Integer (64 Bit mit und ohne Vorzeichen), Flag, String (generisches Byte–Array), IEEE float, Objekte.
- Instruktionen und Daten werden wie Protobuf[4] codiert (7 bits pro Byte, MSB=1 bedeutet „Daten gehen weiter“, höchstwertiger Teil zuerst)
- Vier Stacks: integer, float, objects, strings
- `endwith` und `endcmd` um Objekt–Message–Blöcke und Kommandos zu beenden
- `oswap` um das aktuelle Objekt für's Einfügen in das Vater–Objekt auf den Objekt–Stack zu legen
- `words` für Reflections (die Wörter werden mit Token–Nummer, Identifier und Stack–Effekt gelistet, damit automatische Bindings erzeugbar sind)

Grundlagen

- Fünf Datentypen: Integer (64 Bit mit und ohne Vorzeichen), Flag, String (generisches Byte–Array), IEEE float, Objekte.
- Instruktionen und Daten werden wie Protobuf[4] codiert (7 bits pro Byte, MSB=1 bedeutet „Daten gehen weiter“, höchstwertiger Teil zuerst)
- Vier Stacks: integer, float, objects, strings
- `endwith` und `endcmd` um Objekt–Message–Blöcke und Kommandos zu beenden
- `oswap` um das aktuelle Objekt für's Einfügen in das Vater–Objekt auf den Objekt–Stack zu legen
- `words` für Reflections (die Wörter werden mit Token–Nummer, Identifier und Stack–Effekt gelistet, damit automatische Bindings erzeugbar sind)

Warum binär codieren?

- Schneller und einfacher zu parsen (einfacher bedeutet kleinere Angriffsfläche)
- Es gibt immer noch die Möglichkeit, Kommandos über ein Frontend als Klartext einzugeben
- Der Detokenizer für's Debuggen ist auch sehr einfach
- Der objekt–orientierte Ansatz macht das Schreiben von Anwendungsspezifischer Logik sehr einfach

Warum binär codieren?

- Schneller und einfacher zu parsen (einfacher bedeutet kleinere Angriffsfläche)
- Es gibt immer noch die Möglichkeit, Kommandos über ein Frontend als Klartext einzugeben
- Der Detokenizer für's Debuggen ist auch sehr einfach
- Der objekt–orientierte Ansatz macht das Schreiben von Anwendungsspezifischer Logik sehr einfach

Warum binär codieren?

- Schneller und einfacher zu parsen (einfacher bedeutet kleinere Angriffsfläche)
- Es gibt immer noch die Möglichkeit, Kommandos über ein Frontend als Klartext einzugeben
- Der Detokenizer für's Debuggen ist auch sehr einfach
- Der objekt–orientierte Ansatz macht das Schreiben von Anwendungsspezifischer Logik sehr einfach

Warum binär codieren?

- Schneller und einfacher zu parsen (einfacher bedeutet kleinere Angriffsfläche)
- Es gibt immer noch die Möglichkeit, Kommandos über ein Frontend als Klartext einzugeben
- Der Detokenizer für's Debuggen ist auch sehr einfach
- Der objekt–orientierte Ansatz macht das Schreiben von Anwendungsspezifischer Logik sehr einfach

Warum eine Programmiersprache als Daten?

Lemma: jede glue logic wird irgendwann Turing–complete

- Implementiere nur das, was du brauchst — aber du solltest nicht mehr als einen generischen Interpreter implementieren
- Die typische Vorgehensweise für RPC: Das ganze Objekt serialisieren (mit Unterobjekten), über's Netz senden, und eine Funktion auf das Objekt anwenden
- Das Konzept von Net2o (basiert auf ONF): Halte das Objekt synchron und sende nur die Änderungen — mit einfachen Settern und Gettern, und was man sonst für Funktionen braucht
- Damit kann man viele Aufrufe auf einmal übertragen, und reduziert die Latenz

Warum eine Programmiersprache als Daten?

Lemma: jede glue logic wird irgendwann Turing–complete

- Implementiere nur das, was du brauchst — aber du solltest nicht mehr als einen generischen Interpreter implementieren
- Die typische Vorgehensweise für RPC: Das ganze Objekt serialisieren (mit Unterobjekten), über's Netz senden, und eine Funktion auf das Objekt anwenden
- Das Konzept von Net2o (basiert auf ONF): Halte das Objekt synchron und sende nur die Änderungen — mit einfachen Settern und Gettern, und was man sonst für Funktionen braucht
- Damit kann man viele Aufrufe auf einmal übertragen, und reduziert die Latenz

Warum eine Programmiersprache als Daten?

Lemma: jede glue logic wird irgendwann Turing–complete

- Implementiere nur das, was du brauchst — aber du solltest nicht mehr als einen generischen Interpreter implementieren
- Die typische Vorgehensweise für RPC: Das ganze Objekt serialisieren (mit Unterobjekten), über's Netz senden, und eine Funktion auf das Objekt anwenden
- Das Konzept von Net2o (basiert auf ONF): Halte das Objekt synchron und sende nur die Änderungen — mit einfachen Settern und Gettern, und was man sonst für Funktionen braucht
- Damit kann man viele Aufrufe auf einmal übertragen, und reduziert die Latenz

Warum eine Programmiersprache als Daten?

Lemma: jede glue logic wird irgendwann Turing–complete

- Implementiere nur das, was du brauchst — aber du solltest nicht mehr als einen generischen Interpreter implementieren
- Die typische Vorgehensweise für RPC: Das ganze Objekt serialisieren (mit Unterobjekten), über's Netz senden, und eine Funktion auf das Objekt anwenden
- Das Konzept von Net2o (basiert auf ONF): Halte das Objekt synchron und sende nur die Änderungen — mit einfachen Settern und Gettern, und was man sonst für Funktionen braucht
- Damit kann man viele Aufrufe auf einmal übertragen, und reduziert die Latenz

Sicherheit

Lemma: jedes hinreichend komplexe Format kann missbraucht werden

Also beschränkt man sich auf ein ganz einfaches Format

Interpreter

```
: cmd@ ( -- u )
  buf-state 2@ over + >r p@+ r> over - buf-state 2! 64>n ;
: n>cmd ( n -- addr )  cells >r
  o IF   token-table  ELSE  setup-table  THEN
  $@ r@ u<= IF  net2o-crash  THEN  r> + ;
: cmd-dispatch ( addr u -- addr' u' )  buf-state 2!
  cmd@ n>cmd @ ?dup IF  execute  ELSE  net2o-crash  THEN
  buf-state 2@ ;
: cmd-loop ( addr u -- )
  BEGIN  cmd-dispatch dup 0<= UNTIL  2drop ;
```

Sicherheit

Lemma: jedes hinreichend komplexe Format kann missbraucht werden

Also beschränkt man sich auf ein ganz einfaches Format

Interpreter

```
: cmd@ ( -- u )
  buf-state 2@ over + >r p@+ r> over - buf-state 2! 64>n ;
: n>cmd ( n -- addr )  cells >r
  o IF   token-table  ELSE  setup-table  THEN
  $@ r@ u<= IF  net2o-crash  THEN  r> + ;
: cmd-dispatch ( addr u -- addr' u' )  buf-state 2!
  cmd@ n>cmd @ ?dup IF  execute  ELSE  net2o-crash  THEN
  buf-state 2@ ;
: cmd-loop ( addr u -- )
  BEGIN  cmd-dispatch dup 0<= UNTIL  2drop ;
```

Dateien lesen

lese drei Dateien

```
0 lit, file-id "net2o.fs" $, 0 lit,
open-file get-size get-stat endwith
1 lit, file-id "data/2011-
05-13_11-26-57-small.jpg" $, 0 lit,
open-file get-size get-stat endwith
2 lit, file-id "data/2011-
05-20_17-01-12-small.jpg" $, 0 lit,
open-file get-size get-stat endwith
```

Dateien lesen: Antwort

lese drei Dateien: Antwort

```
0 lit, file-id 12B9A lit, set-size
    2014-
8-24T13:52:27.220Z lit, 1A4 lit, set-stat endwith
1 lit, file-id 9C65C lit, set-size
    2014-
7-27T00:34:15.309Z lit, 1A4 lit, set-stat endwith
2 lit, file-id 9D240 lit, set-size
    2014-
7-27T00:34:15.427Z lit, 1A4 lit, set-stat endwith
```

Messages

messages

```
msg "[: msg-start "Hi Bernd" $, msg-text ;]"+
    "<pubkey>"+"<date-span>"+"<signature>" $,
    nestsig endwith
"<reply-token>" push-$ push' nest 0 ok?
```

Structured Text a la HTML

HTML–like structured text

```
body
   p "Some text with " text
       bold "bold" text oswap add
       " markup" text
   oswap add
   li
       ul "a bullet point" text oswap add
       ul "another bullet point" text oswap add
   oswap add
oswap add
```

For Further Reading I

Bernd Paysan

net2o source repository and wiki

https://fossil.net2o.de/net2o

Health & Safety Executive HSE – UK

Out of control, 2nd edition 2003

http://www.hse.gov.uk/pubns/priced/hsg238.pdf

Martin Croxford and Dr. Roderick Chapman

Correctness by Construction: A Manifesto for High-Integrity Software

http://www.crosstalkonline.org/storage/issue-archives/2005/200512/200512-Croxford.pdf

Google Developers

Protocol Buffers

https://developers.google.com/protocol-buffers/