nftables – Der neue Paketfilter im Linux-Kernel

CLT 2014

15. März 2014

Michael Steinfurth
Linux/Unix Consultant & Trainer
B1 Systems GmbH
steinfurth@b1-systems.de

Agenda

Vorstellung B1 Systems

nftables – Einführung

Funktionsweise

Was kann nftables?

Beispiel – Praxis

Vorstellung B1 Systems

- gegründet 2004
- primär Linux/Open Source-Themen
- national & international tätig
- über 60 Mitarbeiter
- unabhängig von Soft- und Hardware-Herstellern
- Leistungsangebot:
  - Beratung & Consulting
  - Support
  - Entwicklung
  - Training
  - Betrieb
  - Lösungen
- dezentrale Strukturen

Schwerpunkte

- Virtualisierung (XEN, KVM & RHEV)
- Systemmanagement (Spacewalk, Red Hat Satellite, SUSE Manager)
- Konfigurationsmanagement (Puppet & Chef)
- Monitoring (Nagios & Icinga)
- IaaS Cloud (OpenStack & SUSE Cloud)
- Hochverfügbarkeit (Pacemaker)
- Shared Storage (GPFS, OCFS2, DRBD & CEPH)
- Dateiaustausch (ownCloud)
- Paketierung (Open Build Service)
- Administratoren oder Entwickler zur Unterstützung des Teams vor Ort

nftables – Einführung

Was ist nftables?

- moderner Paketfilter im Linux-Kernel
- das Programm im Nutzerbereich (*User Space*)
- zustandsgesteuert
- keine Veränderung am *Netfilter- & Connection-Tracking*-System
- iptables-Ersatz?

Was ist nftables?

- moderner Paketfilter im Linux-Kernel
- das Programm im Nutzerbereich (*User Space*)
- zustandsgesteuert
- keine Veränderung am *Netfilter- & Connection-Tracking*-System
- iptables-Ersatz?

Was ist nftables?

- moderner Paketfilter im Linux-Kernel
- das Programm im Nutzerbereich (*User Space*)
- zustandsgesteuert
- keine Veränderung am *Netfilter- & Connection-Tracking*-System
- iptables-Ersatz?

Was ist nftables?

- moderner Paketfilter im Linux-Kernel
- das Programm im Nutzerbereich (*User Space*)
- zustandsgesteuert
- keine Veränderung am *Netfilter-* & *Connection-Tracking*-System
- iptables-Ersatz?

Was ist nftables?

- moderner Paketfilter im Linux-Kernel
- das Programm im Nutzerbereich (*User Space*)
- zustandsgesteuert
- keine Veränderung am *Netfilter-* & *Connection-Tracking*-System
- iptables-Ersatz?

Was ist nftables?

- moderner Paketfilter im Linux-Kernel
- das Programm im Nutzerbereich (*User Space*)
- zustandsgesteuert
- keine Veränderung am *Netfilter-* & *Connection-Tracking*-System
- iptables-Ersatz?

Ja! wenn ausgereift und stabil

Geschichte

- NFWS 2008: P. McHardy “Nftables: an iptables rewrite”
- 18. März 2009: Linux-netdev “First release of nftables”
- Kommentar am 20. März 2010: *in Arbeit*
- Stillstand bis 2012
- die Firma Astaro sponsort Weiterentwicklung
- Aufnahmeantrag in den Hauptentwicklungszweig
- 19. Januar Release 3.13 Einzug in den Linux-Kernel

Geschichte

- NFWS 2008: P. McHardy “Nftables: an iptables rewrite”
- 18. März 2009: Linux-netdev “First release of nftables”
- Kommentar am 20. März 2010: *in Arbeit*
- Stillstand bis 2012
- die Firma Astaro sponsort Weiterentwicklung
- Aufnahmeantrag in den Hauptentwicklungszweig
- 19. Januar Release 3.13 Einzug in den Linux-Kernel

Geschichte

- NFWS 2008: P. McHardy “Nftables: an iptables rewrite”
- 18. März 2009: Linux-netdev “First release of nftables”
- Kommentar am 20. März 2010: *in Arbeit*
- Stillstand bis 2012
- die Firma Astaro sponsort Weiterentwicklung
- Aufnahmeantrag in den Hauptentwicklungszweig
- 19. Januar Release 3.13 Einzug in den Linux-Kernel

Geschichte

- NFWS 2008: P. McHardy “Nftables: an iptables rewrite”
- 18. März 2009: Linux-netdev “First release of nftables”
- Kommentar am 20. März 2010: *in Arbeit*
- Stillstand bis 2012
- die Firma Astaro sponsort Weiterentwicklung
- Aufnahmeantrag in den Hauptentwicklungszweig
- 19. Januar Release 3.13 Einzug in den Linux-Kernel

Geschichte

- NFWS 2008: P. McHardy “Nftables: an iptables rewrite”
- 18. März 2009: Linux-netdev “First release of nftables”
- Kommentar am 20. März 2010: *in Arbeit*
- Stillstand bis 2012
- die Firma Astaro sponsort Weiterentwicklung
- Aufnahmeantrag in den Hauptentwicklungszweig
- 19. Januar Release 3.13 Einzug in den Linux-Kernel

Geschichte

- NFWS 2008: P. McHardy “Nftables: an iptables rewrite”
- 18. März 2009: Linux-netdev “First release of nftables”
- Kommentar am 20. März 2010: *in Arbeit*
- Stillstand bis 2012
- die Firma Astaro sponsort Weiterentwicklung
- Aufnahmeantrag in den Hauptentwicklungszweig
- 19. Januar Release 3.13 Einzug in den Linux-Kernel

Geschichte

- NFWS 2008: P. McHardy “Nftables: an iptables rewrite”
- 18. März 2009: Linux-netdev “First release of nftables”
- Kommentar am 20. März 2010: *in Arbeit*
- Stillstand bis 2012
- die Firma Astaro sponsort Weiterentwicklung
- Aufnahmeantrag in den Hauptentwicklungszweig
- 19. Januar Release 3.13 Einzug in den Linux-Kernel

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- bisher: Binäraustausch bei Regeländerungen
  - nicht inkrementell, kompletter Regelsatz
  - Kommunikation mit Kernel über raw Sockets
    - ↪ je mehr Regeln, desto höhere Austauschzeit
- Erweiterungen im Userspace & Kernelspace
  - pro Funktion einmal im Kernel und einmal im Userspace
  - Erweiterungen zweifacher Entwicklungsaufwand
  - mehrere Erweiterungen pro Anfrage
    - ↪ mehr Leistungsanforderungen
  - keine “offizielle Bilbiothek”
  - redundanter Quelltext unter verschiedenen Modulen

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in “modernen” Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen
    - ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in “modernen” Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen
    - ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in “modernen” Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen

    ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in "modernen" Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen

    ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in “modernen” Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen
    - ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in "modernen" Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen

    ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in "modernen" Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen

    ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in "modernen" Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen
    - ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

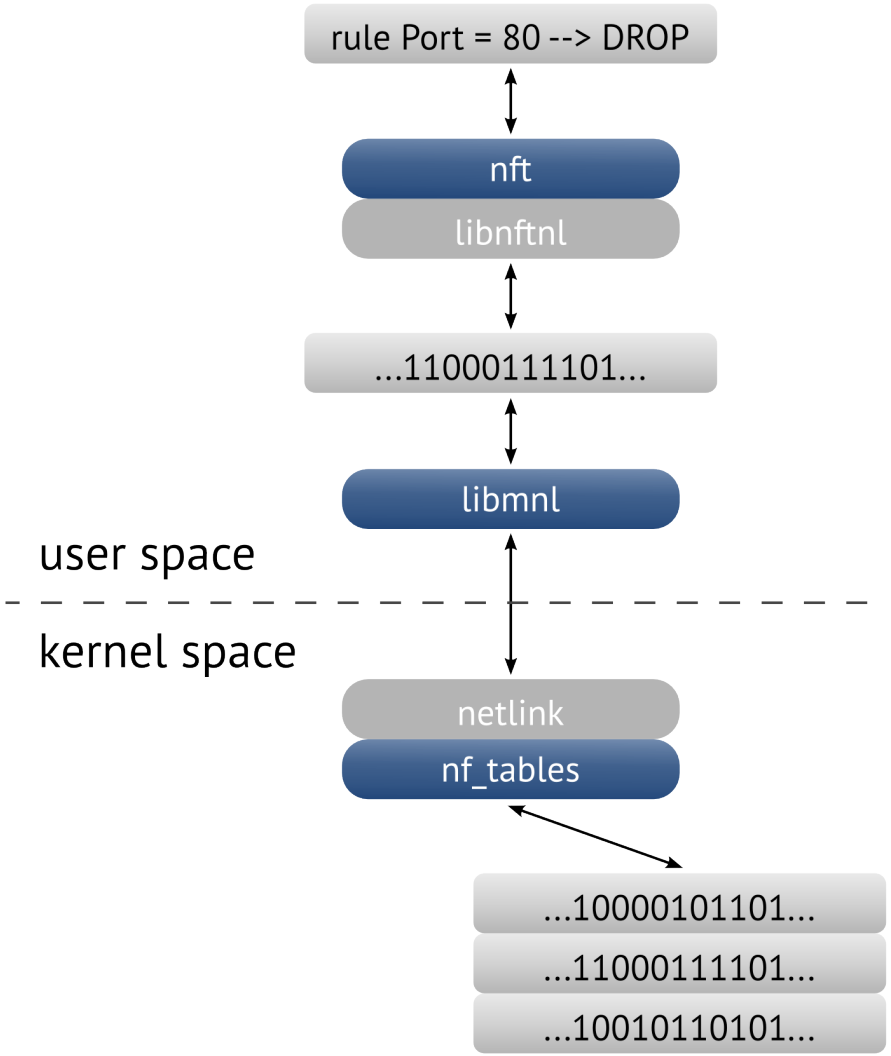
- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in "modernen" Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen
    - ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Warum nftables und nicht mehr iptables?

- leichte Leistungseinbußen auch ohne Regeln
  - geladene Module
  - verbunden mit Netfilter
- eigene Programme für unterschiedliche Protokollebenen
  - ebtables, arptables, iptables, ip6tables
- schwierig in "modernen" Netzen
  - verschiedene IP-Adressen aus verschiedenen Bereichen
  - 1-zu-1 Beschränkungen
    - ↪ viele Regeln
  - ungünstig für wechselnde virtuelle Maschinen (Cloud)

Funktionsweise

Übersicht


rule Port = 80 --> DROP
nft
libnftnl
...11000111101...
libmnl
user space
kernel space
netlink
nf_tables
...10000101101...
...11000111101...
...10010110101...

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

Beispiel `nft`

```
# nft list table filter
```

- Administrator benutzt *nft* zur Steuerung im Kernel
- aktivieren, auslesen, konfigurieren, löschen
- Befehlszeilenprogramm mit Parametern
- Regeln ohne Optionsparameter (`-o`, `--option`)
- Optionen:
  - `[-i]` interaktive
  - `[-f]` von Datei lesen
  - `[-n]` numerisch (mehrfach: nn, nnn)
  - `[-a]` mehr Information (Regel-"Handle")

nft, das Administrationswerkzeug

- Grammatik mit Stichworten/Symbolen (*token*)
- *Scanner* erfasst *TOKEN*
- Zusammenarbeit mit einem *Parser*
- Umsetzung der kontext-freien Grammatik in Funktionen
- *Scanner* in *flex* geschrieben
- *Parser* in *GNU bison*

nft, das Administrationswerkzeug

- Grammatik mit Stichworten/Symbolen (*token*)
- *Scanner* erfasst *TOKEN*
- Zusammenarbeit mit einem *Parser*
- Umsetzung der kontext-freien Grammatik in Funktionen
- *Scanner* in *flex* geschrieben
- *Parser* in *GNU bison*

nft, das Administrationswerkzeug

- Grammatik mit Stichworten/Symbolen (*token*)
- *Scanner* erfasst *TOKEN*
- Zusammenarbeit mit einem *Parser*
- Umsetzung der kontext-freien Grammatik in Funktionen
- *Scanner* in *flex* geschrieben
- *Parser* in *GNU bison*

nft, das Administrationswerkzeug

- Grammatik mit Stichworten/Symbolen (*token*)
- *Scanner* erfasst *TOKEN*
- Zusammenarbeit mit einem *Parser*
- Umsetzung der kontext-freien Grammatik in Funktionen
- *Scanner* in *flex* geschrieben
- *Parser* in *GNU bison*

nft, das Administrationswerkzeug

- Grammatik mit Stichworten/Symbolen (*token*)
- *Scanner* erfasst *TOKEN*
- Zusammenarbeit mit einem *Parser*
- Umsetzung der kontext-freien Grammatik in Funktionen
- *Scanner* in *flex* geschrieben
- *Parser* in *GNU bison*

nft, das Administrationswerkzeug

- Grammatik mit Stichworten/Symbolen (*token*)
- *Scanner* erfasst *TOKEN*
- Zusammenarbeit mit einem *Parser*
- Umsetzung der kontext-freien Grammatik in Funktionen
- *Scanner* in *flex* geschrieben
- *Parser* in *GNU bison*

Übergang libnftnl zum Kernel

- “Verpacken” der Regeln in Netlink-Nachrichten
- binär, keine Text-Stichworte
- libmnl kommuniziert mit Kernel über *nfnetlink*
- Kernel lauscht auf netlink-Nachrichten
- generischen Operationen und Werte aus Nachricht
  - ↪ Zustandsautomat

Übergang libnftnl zum Kernel

- “Verpacken” der Regeln in Netlink-Nachrichten
- binär, keine Text-Stichworte
- libmnl kommuniziert mit Kernel über *nfnetlink*
- Kernel lauscht auf netlink-Nachrichten
- generischen Operationen und Werte aus Nachricht
  - ↪ Zustandsautomat

Übergang libnftnl zum Kernel

- “Verpacken” der Regeln in Netlink-Nachrichten
- binär, keine Text-Stichworte
- libmnl kommuniziert mit Kernel über *nfnetlink*
- Kernel lauscht auf netlink-Nachrichten
- generischen Operationen und Werte aus Nachricht
  - ↪ Zustandsautomat

Übergang libnftnl zum Kernel

- “Verpacken” der Regeln in Netlink-Nachrichten
- binär, keine Text-Stichworte
- libmnl kommuniziert mit Kernel über *nfnetlink*
- Kernel lauscht auf netlink-Nachrichten
- generischen Operationen und Werte aus Nachricht
  - ↪ Zustandsautomat

Übergang libnftnl zum Kernel

- “Verpacken” der Regeln in Netlink-Nachrichten
- binär, keine Text-Stichworte
- libmnl kommuniziert mit Kernel über *nfnetlink*
- Kernel lauscht auf netlink-Nachrichten
- generischen Operationen und Werte aus Nachricht
  - ↪ Zustandsautomat

Übergang libnftnl zum Kernel

- “Verpacken” der Regeln in Netlink-Nachrichten
- binär, keine Text-Stichworte
- libmnl kommuniziert mit Kernel über *nfnetlink*
- Kernel lauscht auf netlink-Nachrichten
- generischen Operationen und Werte aus Nachricht
  - ↪ Zustandsautomat

Zustandsautomat nach BPF

- 4 Register + versch. Filter-Befehlssätze

netlink nachrichten Inhalt

```
ip filter input 0 0
  [ payload load 4b @ network header + 12 => reg 1 ]
  [ cmp eq reg 1 0x0a0a0a0a ]
  [ immediate reg 0 accept ]
```

Zustandsautomat nach BPF

- 4 Register + versch. Filter-Befehlssätze

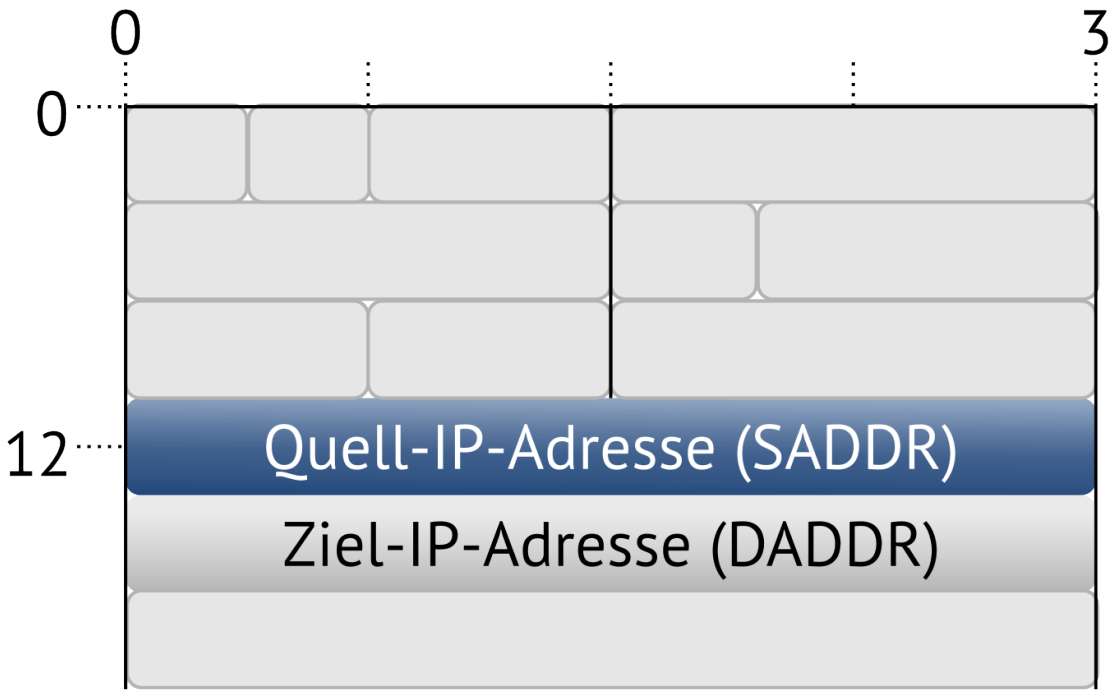
netlink nachrichten Inhalt

```
ip filter input 0 0
  [ payload load 4b @ network header + 12 => reg 1 ]
  [ cmp eq reg 1 0x0a0a0a0a ]
  [ immediate reg 0 accept ]
```

```
nft add rule ip filter input ip saddr 10.10.10.10 accept
```

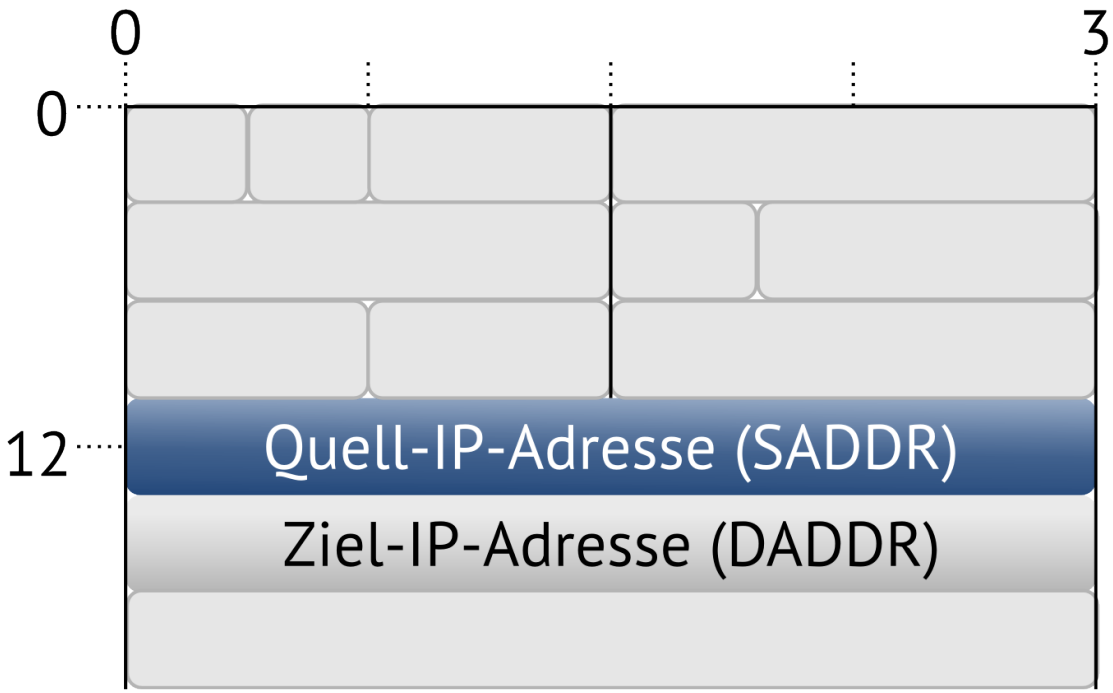
Erklärung payload

```
payload load 4b @ network header + 12
```



Erklärung payload

```
payload load 4b @ network header + 12
```




```
payload load 4b @ network header + 16
```

Was kann nftables?

Grundlegende Funktionen

- Tabellen
- base-Ketten
- Netfilter Hooks
- Ketten
- Regeln

Tabellen

Tabelle erstellen/anzeigen/leeren/löschen

```
# nft add table ip filter
# nft list table ip filter
# nft flush table ip filter
# nft delete table ip filter
```

- Tabellenname frei wählbar
- Protokollarten `ip`,`ip6`,`arp`,`bridge`,`(inet)`
- gibt es nicht automatisch!

base-Ketten

- registriert an `netfilter hooks`
  - `prerouting,input,forward,output,postrouting`
- type-basiert:
  - `filter, nat, route` (ähnl. `mangle`)
- Priorität
- keine “Default Policy”!

Erstellen einer base-Kette

```
# nft add chain \
filter input { type filter hook input priority 0 \; }
```

base-Ketten

- registriert an `netfilter hooks`
  - `prerouting,input,forward,output,postrouting`
- type-basiert:
  - `filter`, `nat`, `route` (ähnl. `mangle`)
- Priorität
- keine “Default Policy”!

Erstellen einer base-Kette

```
# nft add chain \
filter input { type filter hook input priority 0 \; }
```

base-Ketten

- registriert an `netfilter hooks`
  - `prerouting,input,forward,output,postrouting`
- type-basiert:
  - `filter`, `nat`, `route` (ähnl. `mangle`)
- Priorität
- keine “Default Policy”!

Erstellen einer base-Kette

```
# nft add chain \
filter input { type filter hook input priority 0 \; }
```

base-Ketten

- registriert an `netfilter hooks`
  - `prerouting,input,forward,output,postrouting`
- type-basiert:
  - `filter`, `nat`, `route` (ähnl. `mangle`)
- Priorität
- keine “Default Policy”!

Erstellen einer base-Kette

```
# nft add chain \
filter input { type filter hook input priority 0 \; }
```

base-Ketten

- registriert an `netfilter hooks`
  - `prerouting,input,forward,output,postrouting`
- type-basiert:
  - `filter`, `nat`, `route` (ähnl. `mangle`)
- Priorität
- keine “Default Policy”!

Erstellen einer base-Kette

```
# nft add chain \
filter input { type filter hook input priority 0 \; }
```

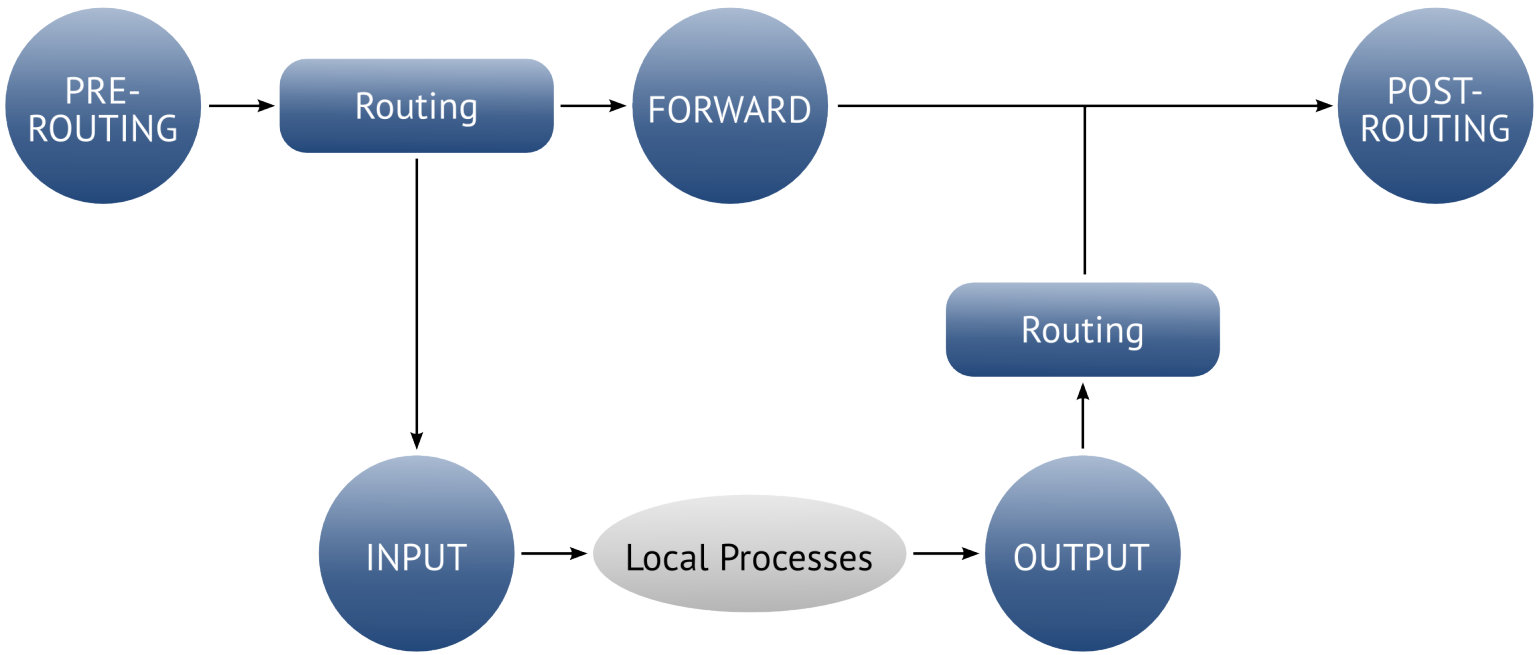
base-Ketten

- registriert an `netfilter hooks`
  - `prerouting,input,forward,output,postrouting`
- type-basiert:
  - `filter`, `nat`, `route` (ähnl. `mangle`)
- Priorität
- keine “Default Policy”!

Erstellen einer base-Kette

```
# nft add chain \
filter input { type filter hook input priority 0 \; }
```

Netfilter Hooks



Ketten

- Nutzer-Ketten (`non-base`) wie in `iptables`
- Kette leeren → löschen
- Ketten löschen → Tabellen löschen

Kette erstellen/leeren/löschen

```
# nft add chain ip filter Kette1
# nft flush chain ip filter Kette1
# nft delete chain ip filter Kette1
```

Ketten

- Nutzer-Ketten (`non-base`) wie in `iptables`
- Kette leeren → löschen
- Ketten löschen → Tabellen löschen

Kette erstellen/leeren/löschen

```
# nft add chain ip filter Kette1
# nft flush chain ip filter Kette1
# nft delete chain ip filter Kette1
```

Ketten

- Nutzer-Ketten (`non-base`) wie in `iptables`
- Kette leeren → löschen
- Ketten löschen → Tabellen löschen

Kette erstellen/leeren/löschen

```
# nft add chain ip filter Kette1
# nft flush chain ip filter Kette1
# nft delete chain ip filter Kette1
```

Regeln

- verschiedene Parameter hintereinander
- Aktionen (`verdict`): `accept`, `counter`, `log`, `jump`, `reject`, `drop`

Regel erstellen und löschen

```
# nft add rule filter input saddr 10.10.10.1 count accept
# nft add rule myfilter output ip saddr 10.10.10.0/24 \
          ip daddr 10.10.20.0/24 log accept
# nft add rule filter input tcp dport http counter
# nft add rule ip filter input saddr 10.10.10.1 jump Kette1
# nft delete rule ip filter input handle 1
```

Regeln

- verschiedene Parameter hintereinander
- Aktionen (`verdict`): `accept`, `counter`, `log`, `jump`, `reject`, `drop`

Regel erstellen und löschen

```
# nft add rule filter input saddr 10.10.10.1 count accept
# nft add rule myfilter output ip saddr 10.10.10.0/24 \
          ip daddr 10.10.20.0/24 log accept
# nft add rule filter input tcp dport http counter
# nft add rule ip filter input saddr 10.10.10.1 jump Kette1
# nft delete rule ip filter input handle 1
```

Allgemeines

Datei: myrules

```
table ip myfilter {
        chain input {
                type filter hook input priority 0;
                ip saddr 10.10.10.1 accept
                ip saddr 10.10.10.0/24 log drop
        }

        chain forward {
                type filter hook forward priority 0;
        }

        chain output {
                type filter hook output priority 0;
        }
}
```

Erweiterte Funktionen

- Sets
- Dictionaries/Maps
- spezielle Maps
- Intervalle

Sets

- Datensätze werden überprüft
- z.B. nicht für jede IP-Adresse eine Regel
- anonyme und namensbasierte
- anonyme Sets nicht veränderbar

Anonyme Sets

```
# nft add rule myfilter input tcp dport { 80, 443 } accept
# nft add rule myfilter input ip saddr \
          { 10.10.10.1, 10.10.20.1 } accept
```

Sets

- namensbasierte Sets können verändert werden
- (mehrfacher) Zugriff über Regeln
- Zugriff durch Regel unterbindet Löschen
- verschieden Typen (IP(6)-Adressen, Ports, Protocol, Marks, IF-Indexe)

Namensbasierte Sets

```
# nft add set myfilter webserver { type ipv4_address\; }
# nft add element myfilter \
          webserver { 10.10.80.1, 10.10.81.2 }
# nft list set myfilter webserver
# nft add rule myfilter input ip daddr @webserver accept
# nft delete set myfilter webserver
```

Sets

- namensbasierte Sets können verändert werden
- (mehrfacher) Zugriff über Regeln
- Zugriff durch Regel unterbindet Löschen
- verschieden Typen (IP(6)-Adressen, Ports, Protocol, Marks, IF-Indexe)

Namensbasierte Sets

```
# nft add set myfilter webserver { type ipv4_address\; }
# nft add element myfilter \
          webserver { 10.10.80.1, 10.10.81.2 }
# nft list set myfilter webserver
# nft add rule myfilter input ip daddr @webserver accept
# nft delete set myfilter webserver
```

Sets

- namensbasierte Sets können verändert werden
- (mehrfacher) Zugriff über Regeln
- Zugriff durch Regel unterbindet Löschen
- verschieden Typen (IP(6)-Adressen, Ports, Protocol, Marks, IF-Indexe)

Namensbasierte Sets

```
# nft add set myfilter webserver { type ipv4_address\; }
# nft add element myfilter \
          webserver { 10.10.80.1, 10.10.81.2 }
# nft list set myfilter webserver
# nft add rule myfilter input ip daddr @webserver accept
# nft delete set myfilter webserver
```

Sets

- namensbasierte Sets können verändert werden
- (mehrfacher) Zugriff über Regeln
- Zugriff durch Regel unterbindet Löschen
- verschieden Typen (IP(6)-Adressen, Ports, Protocol, Marks, IF-Indexe)

Namensbasierte Sets

```
# nft add set myfilter webserver { type ipv4_address\; }
# nft add element myfilter \
          webserver { 10.10.80.1, 10.10.81.2 }
# nft list set myfilter webserver
# nft add rule myfilter input ip daddr @webserver accept
# nft delete set myfilter webserver
```

Dictionaries/Maps

- intern: spezielle Sets
- namensbasiert und anonym

anonym

```
# nft add rule myfilter input ip protocol \
vmap {
 udp : jump udpkette, tcp : jump tcpkette, icmp : accept
}
```

Dictionaries/Maps

- intern: spezielle Sets
- namensbasiert und anonym

anonym

```
# nft add rule myfilter input ip protocol \
vmap {
 udp : jump udpkette, tcp : jump tcpkette, icmp : accept
}
```

Dictionaries/Maps

Befehle

```
# nft add map myfilter \
      v_map { type ipv4_address : verdict\; }
# nft add element myfilter \
      v_map { 10.10.10.20 : drop, 10.10.10.40 : drop }
```

Konfigurationausgabe/Datei

```
map v_map {
      type ipv4_address : verdict
      elements = { 10.10.10.40 : drop, 10.10.10.20 : drop }
}
```

Spezielle Maps

Namensbasierte Konfiguration

```
map nat_map {
      type ifindex : ipv4_address
      elements = { eth0 : 10.10.10.1, eth1 : 10.10.20.2 }
}
map jump_map {
      type ipv4_address : verdict
      elements = { 10.10.10.1 : jump Kette1 }
}
```

Intervalle

- Bereiche, die nicht durch Netzmasken abgedeckt werden
- direkt, indirekt als Liste und in Maps

Intervalle

```
# nft add rule myfilter input ip daddr \
      10.10.10.1-10.10.10.25 drop
# nft add rule myfilter input ip daddr \
      {
        10.10.10.1-10.10.10.25,\
        10.10.10.250-10.10.10.255
      } drop
# nft add rule ip filter forward ip daddr vmap \
      {
        10.10.10.1-10.10.10.25 : jump webserver,
        10.10.10.100-10.10.10.111 : jump dbserver
      } drop
```

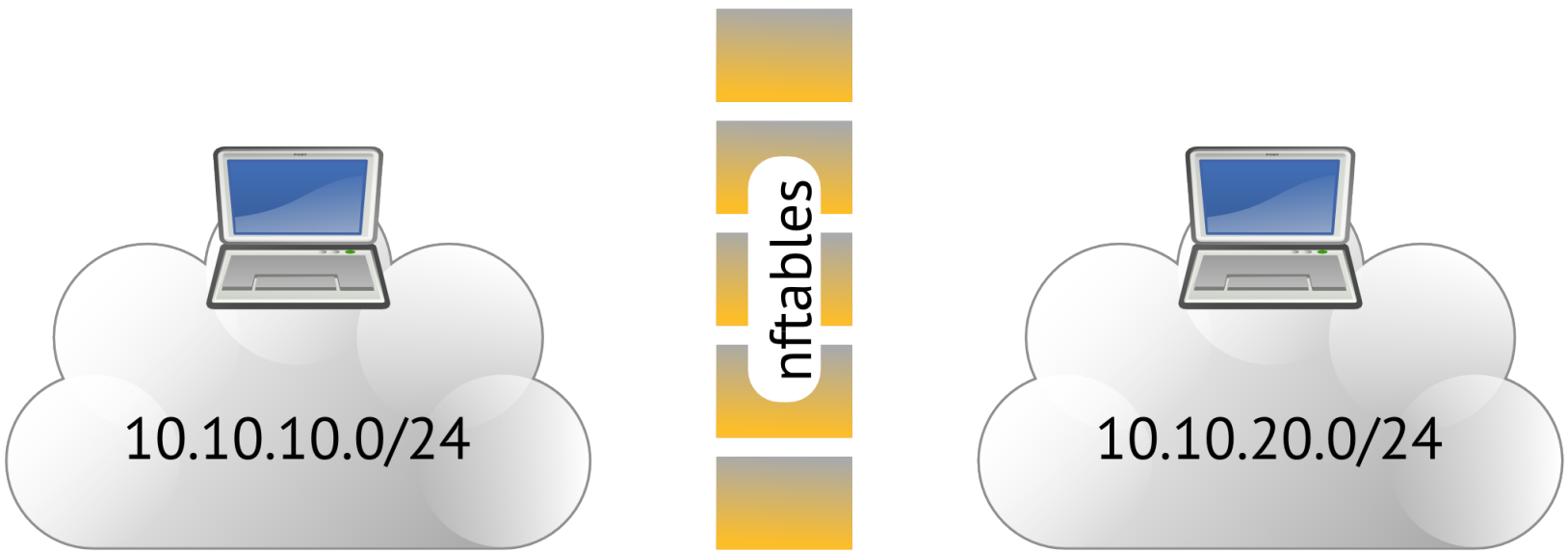
Intervalle

- Bereiche, die nicht durch Netzmasken abgedeckt werden
- direkt, indirekt als Liste und in Maps

Intervalle

```
# nft add rule myfilter input ip daddr \
     10.10.10.1-10.10.10.25 drop
# nft add rule myfilter input ip daddr \
     {
       10.10.10.1-10.10.10.25,\
       10.10.10.250-10.10.10.255
     } drop
# nft add rule ip filter forward ip daddr vmap \
     {
       10.10.10.1-10.10.10.25 : jump webserver,
       10.10.10.100-10.10.10.111 : jump dbserver
     } drop
```

Beispiel – Praxis



Quellen

- `http://lwn.net/Articles/324251/` – Ankündigung nftables
- `http://home.regit.org/*` – Eric Leblond (Core Entwickler)
- `slideshare.net` – Kernel-recipes-2013 – nftables Vortrag – Eric Leblond
- `http://people.netfilter.org/wiki-nftables/`

Vielen Dank für die Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de
oder +49 (0)8457 - 931096