Osaka University

Analysis on Evolution of Network-related Functions in the Linux Kernel

Linux カーネルにおける ネットワーク機能進化の分析

宮川 裕考
情報ネットワーク学専攻 村田研究室

[17] Y. Gao, Z. Zheng, and F. Qin, "Analysis of linux kernel as a complex network," Chaos, Solitons & Fractals, vol. 69, pp. 246 - 252, Nov. 2014.

研究の背景と目的

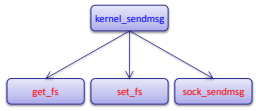
- **ネットワーク機能仮想化 (NFV) が注目**
  - ソフトウェア化したネットワーク機能を 汎用ハードウェア上の仮想マシンで実行
  - 柔軟かつ迅速なサービス展開を可能とすることが期待
  - 現状ではサービスに対する要求を詳細に把握した上での設備設計が前提となるため、ネットワーク機能は固定的に提供
- **ネットワーク機能のコンポーネント化と連携による柔軟性向上**
  - コンポーネントの結合・連携によってネットワーク機能を動的に形成し、より柔軟にサービスを提供
- **ネットワーク機能のソフトウェア実装の様相を分析**
  - Linuxカーネル全体を分析[17]：ソフトウェア品質やバグ予測を目的

Linuxカーネルのネットワーク実装を題材とし、共通して利用される関数群（コア機能）を抽出
コア機能の経年変化を分析し、他の機能との連携の様相やサイズ変化を明らかにする

Linuxカーネルの分析

- **分析対象**
  - Linux カーネル 2.4.0 (2001) – 4.7 (2016)
- **分析手順**
  - Linux カーネルからコールグラフを生成
    - コールグラフ：関数をノード、関数呼び出しをリンクとした有向グラフ
    - 通信に関連した関数群が保持されるディレクトリ'net' を対象として作成
  - コールグラフを構成するノードをコンポーネントに分類
    - 関数単位では細かすぎるためネットワーク機能の振る舞いを分析することが困難
  - コンポーネント間の結びつきをバージョンごとに分析

```
int kernel_sendmsg(struct socket *sock, struct
msghdr *msg, struct kvec *vec, size_t num,
size_t size) {
        mm_segment_t oldfs = get_fs();
        int result;
        set_fs(KERNEL_DS);
        // some codes
        result = sock_sendmsg(sock, msg, size);
        set_fs(oldfs);
        return result;}
```



コンポーネントへの分類

- **同じディレクトリに含まれる関数は 同じコンポーネントに所属するよう分類**
  - Linux カーネルのソースファイルは機能ごとにディレクトリ分けされた構造
  - ディレクトリ "net" 直下のソースファイルにおいて宣言された関数についてはコンポーネント"net" に所属
- **例: Linux カーネル 2.4 におけるコンポーネント**

802	appletalk	ax25	bridge	core
ethernet	ipv4	ipx	irda	net
netlink	netrom	packet	rose	sched
sunrpc	unix			

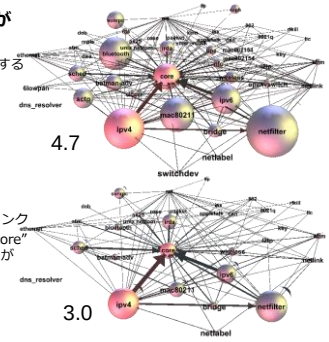
コンポーネント間の接続関係

- **コンポーネントはモジュール構造**
  - 自分自身への呼び出しが多く、他のコンポーネントへの呼び出しは少ない
- **コンポーネント"core"は汎用的な機能を提供**
  - すべてのコンポーネントから利用されている
  - "core"内の関数例：sock_cmsg_send・sock_wmalloc・sock_alloc_send

	net	802	appletalk	ax25	bridge	core	ethernet	ipv4	ipx	irda	netlink	netrom	packet	rose	sched	sunrpc	unix
net	65	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
802	0	2	0	0	0	5	0	1	0	0	0	0	0	0	0	0	0
appletalk	2	4	63	0	0	51	0	0	0	0	0	0	0	0	0	0	0
ax25	2	0	0	292	0	61	0	4	0	0	0	0	0	0	0	0	0
bridge	0	0	0	0	118	23	1	0	0	0	0	0	0	0	0	0	0
core	4	0	0	0	0	162	0	1	0	0	6	0	0	0	7	0	0
ethernet	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
ipv4	5	0	0	0	0	382	0	697	0	0	7	0	0	0	0	0	0
ipx	2	6	0	0	0	40	2	0	52	0	0	0	0	0	0	0	0
irda	2	0	0	0	0	207	1	2	0	986	0	0	0	0	0	0	0
netlink	2	0	0	0	0	26	0	0	0	0	20	0	0	0	0	0	0
netrom	2	0	0	22	0	52	0	2	0	0	0	129	0	0	0	0	0
packet	0	0	0	0	0	49	0	0	0	0	0	0	5	0	0	0	0
rose	2	0	0	28	0	58	0	2	0	0	0	0	0	157	0	0	0
sched	0	0	0	0	0	9	0	0	0	0	0	0	0	0	4	0	0
sunrpc	16	0	0	0	0	8	0	0	0	0	0	0	0	0	0	192	0
unix	3	0	0	0	0	37	0	0	0	0	0	0	0	0	0	0	38

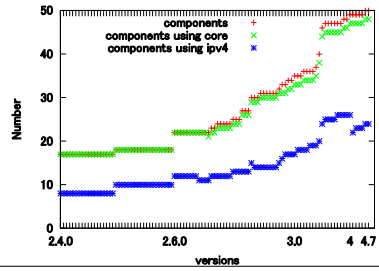
コンポーネントとその接続関係の変化

- **多くのコンポーネントが 追加**
  - 無線やセキュリティに関連するものが多数追加
  - 新しい通信プロトコル
    - SCTP
    - MPLS
- **"core"に加え ipv4が盛んに利用**
  - コンポーネント間リンクの77%が"core"とipv4へのリンク
  - 96%のコンポーネントが"core"を、48%のコンポーネントがipv4を利用
  - ipv4も"core"を利用



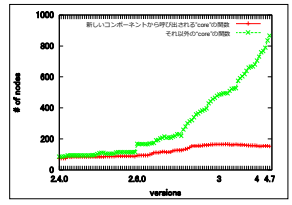
“core”とipv4を利用するコンポーネントの数

- **コンポーネント数の増加に応じて、“core”とipv4を利用するコンポーネントが増加**
  - コンポーネントが追加される際、常に”core”とipv4を利用



“core”の中のコア機能

- **新しいコンポーネントに対して “core” 内の特定の関数で対応**
  - 新しくコンポーネントから “core” への呼び出しの99%において既存の関数が呼び出し先
  - 呼び出し先となっている関数は”core”全体の約15% (version 4.7)
- **“core" の中でも一部がコア機能**
  - 新しいコンポーネントから呼び出される関数の数は開発が進行しても増加しない



まとめと今後の課題

- **まとめ**
  - Linux カーネルからコールグラフを生成し、コンポーネント間の関係を分析
  - coreとipv4がコア機能
    - 多くのコンポーネントが依存
    - 新しく追加されるコンポーネントのほとんどが利用
  - ipv4はcoreに強く依存しており、分散して配置することは不適当
  - core の中でも一部の関数群のみがコア機能
- **今後の課題**
  - 呼び出し頻度を考慮した分析
  - 実際にLinuxカーネルが動作している状況における分析