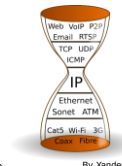
Osaka University

Linuxカーネルを用いたネットワーク機能の接続関係の分析

宮川裕考 荒川伸一 村田正幸
大阪大学 大学院情報科学研究科

インターネットの現状

- **利用形態の多様化**
  - スマートフォンやタブレット端末の普及
  - これまでにない新しいサービスへの期待
- **砂時計型のプロトコルスタック**
  - IP や UDP / TCP が固定的に利用
  - 上位層や下位層のプロトコルの開発が容易
  - 新たな中間層プロトコルによる置き換えが困難
    - あらゆるプロトコルが IP を利用せざるをえない



By Xander89 and OlivierMehani

- **ネットワーク仮想化技術への期待**
  - ネットワーク機能をコンポーネントとして抽出し、仮想化技術を用いてそれらの機能を組み合わせて実行する NFV (Network Function Virtualization) の標準化が進められている
  - プロトコルスタックに束縛されずサービスを提供できる

ネットワーク仮想化技術の課題

- **コンポーネント化の労力が未知**
  - 仮想化したいネットワーク機能が、他のプロトコルやプロトコル内における他のネットワーク機能にどの程度依存しているか
  - ネットワーク機能間の機能分担が適切でない場合、コンポーネント化の際に、多数のネットワーク機能同士の依存関係を鑑みる必要がある
  - ネットワーク機能は、インターネットの利用形態が多様化していくなかで、どのように変化していくか知る必要がある

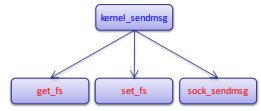
Linuxカーネルのプロトコル実装を題材とし、プロトコル間／プロトコル内のネットワーク機能がどのように結合し、それがどのように変化しているのかを明らかにする

関連研究

- **Linux カーネル全体の関数呼び出し関係を調べた研究 [4]**
  - 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
  - ネットワーク分析手法を適用し、いくつかの統計的指標の推移を調査
  - ソフトウェア品質管理の観点からの機能故障に対する堅牢性を調べている
  - 関数の機能は無視している

```
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;}
```

ソースコード

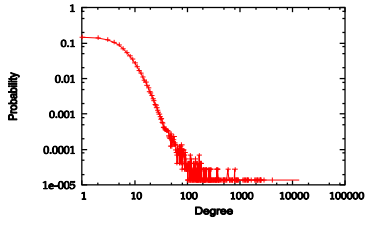


左のソースコードから生成されるコールグラフ

[4] 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.

Linuxカーネルから生成したコールグラフの構造的特徴

- **一部のノードの次数が非常に大きく、大多数のノードの次数は小さい**



コールグラフにおける高次数ノード

- **入出力や同期、メモリ管理など汎用的に利用される関数の次数が高い**

関数名	次数
printk	18707
__builtin_expect	17912
kfree	8417
mutex_unlock	5867
spinlock_check	5762
mutex_lock	5331
memset	5318
memcpy	4999

- **研究目的を達成するには、Linuxカーネル全体ではなく、ネットワーク機能に関連した関数のみに着目する必要がある**

研究のアプローチ

- **対象**
  - Linux カーネル 2.4.0 (2001) – 3.16.7 (2014)
    - 多数の通信プロトコルを実装
- **手法**
  - Linux カーネルから関数呼び出し関係を表す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ネットワーク機能にもとづき分類
    - ネットワーク機能は複数の関数により構成
  - 開発進行にともなう通信関連の関数呼び出し関係の変遷を分析

ネットワーク機能に関連した関数群のコールグラフ

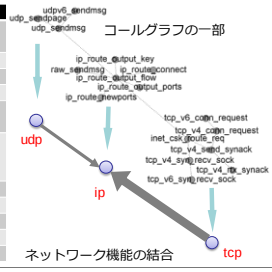
- **通信関連の関数が格納されたディレクトリ`net’からコールグラフを作成**
- **ネットワーク機能に関連した関数群の高次数ノード**
  - 汎用的な関数を除外できている

関数名	次数
netdev_open	81
init_one	75
e1000_probe	74
netdev_close	59
ixgbe_probe	56
hci_cmd_complete_evt	56
ieee80211_tx_status	51

ネットワーク機能にもとづくノードの分類

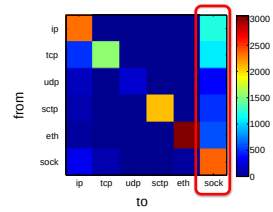
- **コールグラフに含まれるノードをそれが担うネットワーク機能により分類する**
  - ネットワーク機能は関数名に含まれる特定文字列にもとづき分類

分類	対応する正規表現
ip	ip(6|w*|4|6))? inet(4|6)?
tcp	tcp(v?(4|6))?
udp	udp(4|v?6|lite)?
sctp	sctp(v6|probe)?
socket	sock skb?
ethernet	.*80211 arp eth(er|tool)?
icmp	icmp(v(4|6))?
netfilter	nf
nlmsg	(ge)?nlmsg
router	rt(6|nl|nh|netlink|msg|m)?
xfrm	xfrm



ネットワーク機能間における接続関係の分析

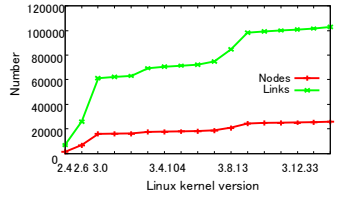
- **ネットワーク機能内のリンクが多い**
- **socket へのリンクが多い**



バージョン 3.16 における機能間のリンク数

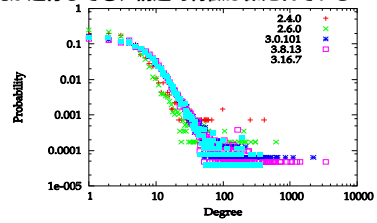
ノード数及びリンク数の推移

- **Linuxカーネルから生成したコールグラフのノード数及びリンク数は開発進行に伴い単調増加している**
- **Linuxカーネルの開発においては、原則として古い機能が削除されることなく機能が追加される**
  - 例えば、現在ではほとんど使われていないネットワーク層プロトコルである IPX の関数も残されている



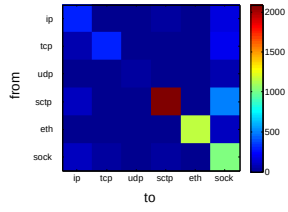
次数分布の変遷

- **3系列間で大きな違いはない**
- **2.4.0及び2.6.0では次数の大きいノードが存在しない**
  - コールグラフの規模が3系列に比べて小さい
- **開発が進行しても、構造的特徴は保たれている**



ネットワーク機能間における接続関係の変遷

- **開発進行により新しいプロトコル (SCTP) へのリンクが急増**
  - SCTP: 比較的新しいトランスポート層プロトコル
- **機能内のリンク（対角線上）の増加が顕著**



バージョン 3.0 から 3.16 までに増加した機能間のリンク数

ネットワーク機能同士の依存度の分析

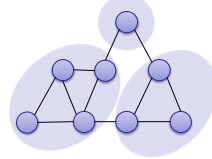
- **評価指標：モジュラリティ**
  - コールグラフをいくつかのノード集合（モジュール）に分割
  - 分割 P に対するモジュラリティ$Q(P)$は以下の式により定義され

$$Q(P) = \frac{1}{2m}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[A_{i,j} - \frac{k_i k_j}{2m}\right]\delta(C_i, C_j)$$

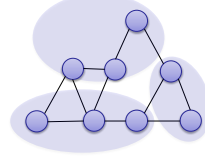
A: グラフの隣接行列 n: ノード数 k: 次数 m: 総リンク数

分割 P において、ノードi とノードj が同じモジュールに属する時1, そうでない時0となる2値変数

  - $Q(P)$が大きいと依存度が小さく、 $Q(P)$が小さいと依存度が大きい



モジュラリティが大きい場合

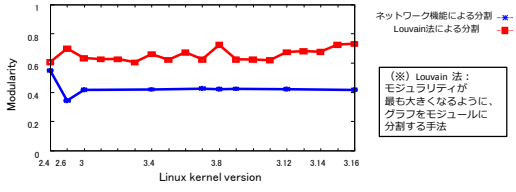


モジュラリティが小さい場合

プロトコル間におけるネットワーク機能同士の依存度

- **ネットワーク機能に基づく分割 Pを与えた場合のモジュラリティを算出**
- **その値はLinuxカーネル 2.6 以降は小さく、ネットワーク機能間の依存度が高い**

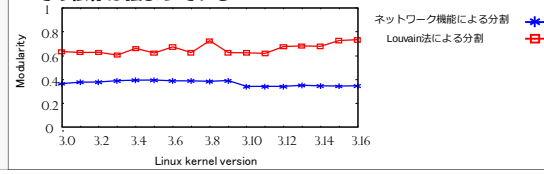
→コンポーネント化の労力が大きい



（※）Louvain 法：モジュラリティが最も大きくなるように、グラフをモジュールに分割する手法

プロトコル内におけるネットワーク機能同士の依存度

- **プロトコル内におけるネットワーク機能同士の依存度を評価**
  - 評価指標: モジュラリティ
  - あるネットワーク機能を担うノードからなるモジュールを、関数名の先頭部分を用いて、さらに分割
  - IP について評価
    - 固定的に利用されるプロトコルであり、コンポーネント化する際のコストが最も懸念
- **プロトコル内においてもネットワーク機能同士の依存度は高く、その依存は強まっている**



まとめと今後の課題

- **まとめ**
  - Linux カーネルのコールグラフを用いて、ネットワーク機能間の関係を分析した
  - プロトコル間とプロトコル内の双方において、ネットワーク機能同士の依存度が高く、その傾向は開発が進むにつれてわずかではあるが強まることが明らかになった

    →ネットワーク機能をコンポーネント化するにあたりコストが増大する

- **今後の課題**
  - ネットワーク機能の使用頻度にもとづいた分析
  - インターネットプロトコルの変化過程の分析
    - ネットワーク仮想化技術を効果的に導入するための方策の提言
    - プロトコルスタック構成法の提案