社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# 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 〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †{h-miyakawa,arakawa,murata}@ist.osaka-u.ac.jp

**あらまし**　現在のインターネットのプロトコルスタックは砂時計型であると言われており、IP および TCP / UDP が固定的に利用されており、新たなサービスの展開が阻害される状況にある。一方で、ネットワークに仮想化機能を導入し、ネットワーク機能をコンポーネント化することにより柔軟なサービス提供を可能とする Network Function Virtualization や Software Defined Networking などの方策も検討されつつあるが、ネットワーク機能のコンポーネント化にあたっては、プロトコル間及びプロトコルを構成する機能群の間で適切に機能分担がなされていることが重要となる。そこで、本稿では、Linux カーネルにおけるインターネットプロトコルスタックの実装を題材として、プロトコル間及びプロトコル内のネットワーク機能間の結合とそれらのカーネル開発に伴う変遷を明らかにすることで、ネットワーク機能を適切にコンポーネント化するための知見を得る。Linux カーネル 2.4.0 から 3.6.17 におけるネットワーク関連の関数の呼び出し関係に着目して分析した結果、機能間の独立性が高くなく、ネットワーク機能のコンポーネント化は困難であることがわかった。

**キーワード**　インターネット、ネットワークプロトコル、プロトコルスタック、砂時計型、Linux カーネル、ネットワーク分析、機能進化

# An analysis of calling structure of network-related functions in Linux kernels

Hirotaka MIYAKAWA†, Shin'ichi ARAKAWA†, and Masayuki MURATA†

† Graduate School of Information Science and Technology, Osaka University
1–5 Yamadaoka, Suita, Osaka 565–0871, Japan
E-mail: †{h-miyakawa,arakawa,murata}@ist.osaka-u.ac.jp

**Abstract**　Recently, network function virtualization has been focused to achieve flexible service deployment. A key to develop the network function virtualization is how easily a network function(s) can be separated from other network functions. In this paper, using the Internet protocol stack in the Linux kernel, we clarify an evolution of functional connectivity in protocols and between protocols through the Linux kernel development. As a result, we found that number of functions and number of function calls have been increased, but degree distribution and path length distribution have not been changed. Moreover, a modularity that indicates independency of network functions is not so high comparing with other biological systems.

**Key words**　Internet, Network Protocol, Protocol Stack, Hour-glass Architecture, Linux Kernel, Network Analysis, Function Evolution

## 1. まえがき

スマートフォンやタブレット端末が普及するとともに、インターネットはますます人々に身近な存在となっている。インターネットを通して提供されるサービスは人々の要求に応じて多様化しており、これまでになかった新しいサービスへの期待が高まっている。その一方で、IP や TCP/UDP などのインターネットの基幹となるネットワークプロトコル群は提供する機能が変わらないまま利用され続けている。

アプリケーションプロトコルやネットワークプロトコルを含むインターネットのプロトコルスタックは砂時計型と言われており [1]、砂時計のくびれとなるネットワーク層とトランスポート層では、IP や TCP/UDP が固定的に用いられ、他のプロトコルはほとんど使用されなくなっている。プロトコルスタックが砂時計型であると、上位層や下位層のプロトコルを開発する際に中間層で対応すべきプロトコル数が少なくて済む。その一方で、新たに開発される上位層や下位層のプロトコルの機能は、固定化した中間層のプロトコルに依拠したものとなる。このため新たな中間層のプロトコルが開発されたとしても置き換えていくことが難しく、あらゆるプロトコルが IP を利用せざるをえない状況にある。

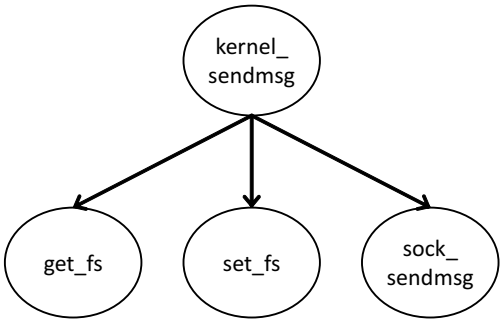
既存のプロトコルの制約から逃れる為に、インターネットのプロトコルスタックに束縛されずサービスを提供できる NFV や SDN といったネットワーク仮想化機能の検討が進められている [2–6]。NFV 等においてネットワーク機能をコンポーネント化するにあたり、プロトコル間及びプロトコル内の機能分担が適切になされていることが重要となる。このためにはインターネットプロトコルスタックがどのように構成されているか、また、それがどのように変化してきたかを明らかにしておく必要がある。適切な機能分担がなされていない場合には、ネットワーク機能をコンポーネント化するにあって多数のプロトコルに改変を加えなければならない。プロトコル間およびプロトコル内の機能依存性が高い場合、つまり他のネットワーク機能に大きく依存している場合には、他のプロトコル内におけるネットワーク機能同士の依存関係を鑑みる必要があり、コンポーネント化のコストが増大する。またプロトコルが担うネットワーク機能は、インターネットの利用が多様化していく中で多種多様に変化しており、プロトコル内のネットワーク機能の依存関係を明らかにすることで、コンポーネント化するに適したネットワーク機能を選定する一助となると考えられる。本稿では、Linux カーネル [7] におけるインターネットプロトコルスタックの実装を題材として、プロトコル間及びプロトコル内の機能結合とその進化の様相を明らかにすることで、ネットワーク機能をコンポーネント化する上での知見を得る。

本稿の内容は以下のとおりである。まず 2 章では、Linux カーネルからコールグラフを生成し、それを分析する。続く 3 章では、生成したコールグラフをネットワーク機能にもとづき分析し、ネットワーク機能の接続構造を明らかにし、4 章ではネットワーク機能の接続構造の変化を分析する。最後に 5 章で本稿のまとめと今後の課題について述べる。

```
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);
    msg->msg_iov = (struct iovec *)vec;
    msg->msg_iovlen = num;
    result = sock_sendmsg(sock, msg, size);
    set_fs(oldfs);
    return result;
}
```

(a) 関数 kernel_sendmsg のプログラムコード




(b) 関数 kernel_sendmsg のコールグラフ

図 1: 関数 kernel_sendmsg のプログラムコードと生成されるコールグラフ

## 2. Linux カーネルから生成されるコールグラフ

プロトコルスタックは、各プロトコルが他のプロトコルを利用する関係を階層的に示したプロトコル群である。そのため、プロトコルスタックの変遷を明らかにするためには、プロトコルを含めたネットワーク機能間の利用関係の変遷を明らかにする必要がある。そこで、各ネットワーク機能間の利用関係を表す、ネットワーク機能の接続構造を分析する。ここでは、ネットワーク機能の接続構造をコールグラフから分析する。ここで接続構造とは、ネットワーク機能同士の利用のされ方を示すものである。本稿では、これを基にしてネットワーク機能を担う関数群の呼び出し関係を分析することにより、ネットワーク機能の接続構造を明らかにする。

### 2.1 関数呼び出し関係にもとづくコールグラフ

コールグラフは関数をノードとし、関数呼び出しをエッジとする有向グラフである。ある関数が別の関数を呼び出している場合、この二つの関数を繋ぐ辺の向きは、呼び出しもとの関数

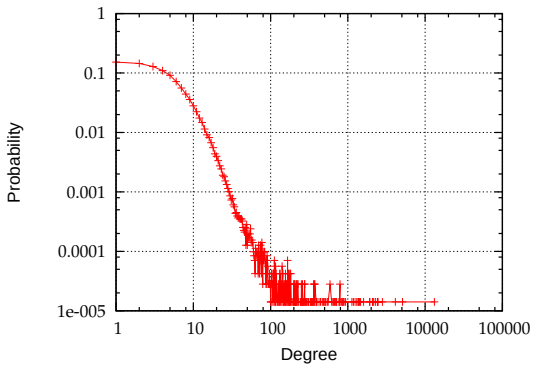



図 2: Linux カーネル 2.6.27 のコールグラフの次数分布

を表すノードから呼び出し先の関数を表すノードである。

### 2.2 Linux カーネルのコールグラフ

コールグラフは CodeViz [8] を使用して作成する。CodeViz は特定バージョンの GCC に対しパッチをあてることにより動作し、コンパイルの際に関数の呼び出し関係を記述したファイル (cdepn ファイル) を生成する。cdepn ファイルはソースファイル毎に生成され、それらのファイルを CodeViz に付属する genfull コマンドにより単一の graph ファイルとする。そこで Linux カーネルを CodeViz のパッチが適用された GCC を用いてコンパイルすることによりコールグラフを生成する。

コールグラフを得ることで、いくつかのネットワーク分析手法を適用し、関数呼び出し関係にあらわれる性質を見る事ができる。例えば、図 2 は Linux カーネル 2.6.27 から上述の手順により得たコールグラフの次数分布を示したものである。これを見ると次数の分布はべき則 [9] に従っていることがわかる。すなわち、一部のノードの次数が大きく、また、多数のノードは次数が小さい性質を有していると言える。しかし、Linux カーネル全体のコールグラフに対する性質を見るのみでは、プロトコルスタックの構成やネットワーク機能の依存関係を見るには不十分である。次数が大きいノードの関数名とその次数を表 1 に示す。この表を見ると、次数が大きいノードは printk や kfree などの汎用的に利用されると推測される関数である。すなわち、ネットワーク機能間及び機能内の結合を理解するには、カーネル全体ではなく、ネットワーク機能に着目した上で分析を行う必要がある。

Linux カーネルの関数呼び出し関係を分析した研究としては、文献 [10, 11] がある。文献 [10] では、Linux カーネル全体の関数呼び出し関係で構成されるネットワークと大腸菌の遺伝子発現制御ネットワークと比較しており、Linux カーネルでは関数の再利用が促進されていることを示しているが、分析対象はプロトコルスタックではなく Linux カーネル全体である。また、文献 [11] では Linux カーネルからコールグラフを作成し、次数分布や平均パス長などの指標を用いて Linux カーネルの変遷を分析しているが、これも Linux カーネル全体を対象としたものである。いずれもコールグラフにおいてハブノードとなっている汎用的な関数を含めて分析を行っている。

表 1: Linux カーネルにおける高次数ノードの例

| 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 |
| _raw_spin_lock_irqsave | 4901 |
| spin_unlock_irqrestore | 4723 |
| __builtin_unreachable | 4029 |
| __builtin_constant_p | 3953 |
| get_current | 3771 |
| spin_unlock | 3602 |
| spin_lock | 3527 |

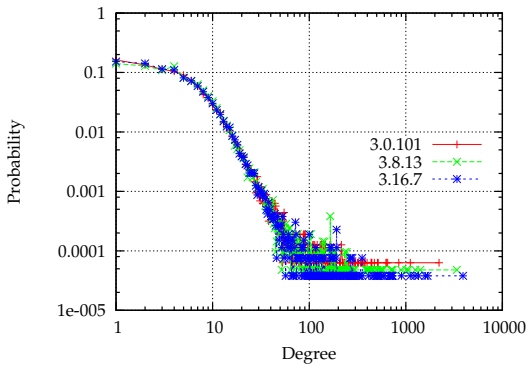



図 3: ディレクトリ net 配下の関数群からなるコールグラフの次数分布

## 3. Linux カーネルにおけるネットワーク機能の接続構造の分析

### 3.1 ネットワーク機能に関連する関数群からなるコールグラフ

コールグラフには Linux カーネルで宣言されたすべての関数が含まれるが、本稿ではネットワーク機能の接続構造に着目しているため、ネットワーク機能に関連した関数のみからなるコールグラフを作成し、分析を行った。

Linux カーネルのソースコードには OS の各種機能により分類されたディレクトリがあり、その中にはネットワーク機能に関連する関数群がまとめられたディレクトリ net がある。例えばディレクトリ net 配下には ethernet や ipv4 といったサブディレクトリがある。そこで、ディレクトリ net 配下の関数を取り出すことにより通信に関連した関数を抽出する。またディレクトリ net 配下の関数がディレクトリ net 配下以外の場所で宣言されている関数を呼び出す場合も分析の対象に含めた。

表 2 にネットワーク機能を担う関数群からなるコールグラフにおける次数上位の関数を示している。これをみると、表 1 でみられた汎用的に用いられた関数が除外できていることがわか

表 2: ネットワーク機能を担う関数群からなるコールグラフにおける高次数ノードの例

| 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 |
| il4965_pci_probe | 53 |
| bond_enslave | 52 |
| ieee80211_tx_status | 51 |
| igb_probe | 50 |
| dev_ethtool | 50 |
| rtl_init_one | 49 |
| il3945_pci_probe | 49 |
| inet6_init | 48 |
| ieee80211_do_stop | 48 |

表 3: ネットワーク機能グループの分類

| グループ | 対応する正規表現 |
|---|---|
| ip | ip(\w*(4\|6))? |
| | inet(4\|6)? |
| tcp | tcp(v?(4\|6))? |
| udp | udp(4\|v?6\|lite)? |
| sctp | sctp(v6—probe)? |
| socket | sock |
| | skb? |
| destination cache | dst |
| network level authentication | nla |
| 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 |

る。図 3 は net 配下の関数群によるコールグラフの次数分布を示している。次数分布はカーネル全体と同じようにべき則に従っていることがわかる。

### 3.2 ネットワーク機能にもとづく関数の分類

関数呼び出しに基いてネットワーク機能の接続構造を明らかにするためには、関数がどのネットワーク機能を担っているのかを分類する必要がある。本稿では、各関数が担うネットワーク機能は関数名により判別する。関数名はアンダースコア (_) を区切り文字として単語が連結されたものが多い。そこで、関数名をアンダースコアで区切り、区切られた文字列に含まれる特定の文字により、関数を各ネットワーク機能に分類する。例えば、ip_tables_init・get_ip_src といった関数はいずれも IP のネットワーク機能に分類する。ネットワーク機能とそのネットワーク機能に分類される正規表現の対応を表 3 に示す。

表 4: Linux カーネル 3.16 における各ネットワーク機能から呼び出されるネットワーク機能

| ネットワーク機能 | 呼び出し回数の多いネットワーク機能 | | |
|---|---|---|---|
| | 最も多い | 2 番目に多い | 3 番目に多い |
| ip | ip | socket | nla |
| tcp | tcp | socket | ip |
| udp | socket | udp | ip |
| icmp | socket | ip | icmp |
| ethernet | ethernet | socket | nla |
| socket | socket | ip | tcp |
| nla | nla | socket | ip |
| dst | dst | socket | ip |
| nlmsg | nlmsg | nla | socket |
| netfilter | netfilter | socket | ip |
| xfrm | xfrm | socket | dst |
| router | router | nla | socket |

### 3.3 ネットワーク機能の接続構造の分析

通信に関連した関数からなるコールグラフに基づき、ネットワーク機能同士の接続構造を分析する。以下では Linux カーネル 3.16 を対象に行なった結果を示す。

各ネットワーク機能を担う関数群間のリンク数を表 4 に示す。表 4 より、同じネットワーク機能を担う関数の間にリンクが多いことがわかる。また IP/TCP の機能に含まれる関数へのリンクが多く、プロトコルスタック全体がそれらのプロトコルに強く依存していることが伺える。

リンク数の多いネットワーク機能に着目すると、ネットワーク機能内で密に、ネットワーク機能間で疎になっているようにみえる。しかし、ネットワーク機能をモジュールとみなした時のモジュール度を計算してみると、その値は約 0.26 と Louvain 法 [12] によりモジュール分割を行った際の値約 0.73 に比べて低いものであった。グラフとしてはモジュール間の依存度が小さいものになっているものの、ネットワーク機能に着目してみるとそうはなっておらず、他のネットワーク機能への依存がみられることがわかる。

## 4. インターネットプロトコルスタックの変遷分析

2001 年 1 月にリリースされたバージョン 2.4.0 から、2014 年 10 月にリリースされたバージョン 3.16.7 までの Linux カーネルを対象とし、コールグラフの構造的特徴及びネットワーク機能の接続構造の変化を示す。

### 4.1 コールグラフの構造的特徴

図 4 は通信に関連した関数からなるコールグラフのノード数およびリンク数の推移を示している。図 4 から開発が進むにしたがってノード数・リンク数が増加していることがわかる。これは、Linux カーネルの開発においては、基本的には古い機能を削除することなく機能が追加されているため、単調増加の傾向にあると考えられる。特に 3.7.10 から 3.9.11 にかけては、ノード数が 18,758 から 24,240、リンク数が 74,869 から 98,279 と急激に増えている。これは、比較的新しく検討された

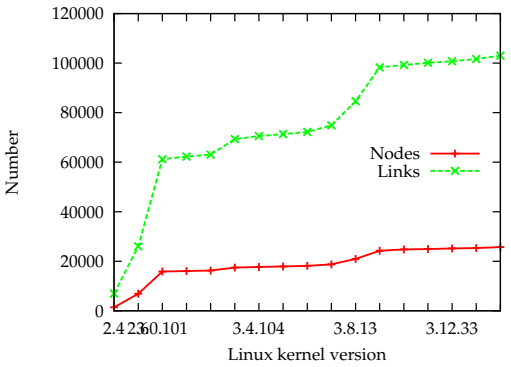



図 4: ノード数およびリンク数の推移

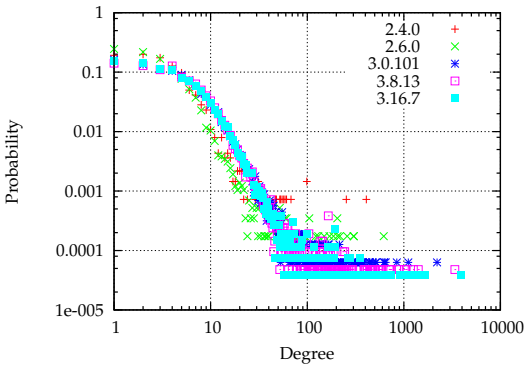



図 5: 次数分布の推移

Stream Control Transmission Protocol (SCTP) [13] の導入が進んだためと考えられる。図 5 は Linux カーネル 2.4.0, 2.6.0, 3.0.101、3.8.13、3.16.7 それぞれのコールグラフにおける次数分布を示している。なお、ここでは次数分布はコールグラフを無向グラフとして算出している。これは predecessor ノードも successor ノードもそのノードが表す関数に関係しているためである。図 5 より、バージョン 3 系列間では大きな違いはないものの、バージョン 2.4.0 及び 2.6.0 は次数の大きいノードの割合が小さくなっている。これは、バージョン 3 系列と比較してコールグラフに含まれるノードが少ないためと考えられる。図 6 は Linux カーネル 2.4.0、2.6.0、3.0.101、3.8.13、3.16.7 それぞれのコールグラフにおけるパス長の分布を示している。ここで言うパス長とは無向グラフにおける最短経路のホップ長であり、Linux カーネルにおける関数呼び出しの深さを表している。パス長も次数分布と同様にコールグラフを無向グラフとして計算している。図 6 より、バージョン間にパス長に大きな変化はなく、開発が進んでも構造的特徴が維持されていることがわかる。

### 4.2 ネットワーク機能間の接続構造の変遷

表 5 に 3.0 から 3.16 にかけてのネットワーク機能間のリンク数の増分の総増加リンク数に占める割合を示している。表 5 より機能内のリンク増加が顕著であることがわかる。次にネットワーク機能間のつながりの疎密がどのように変化しているかを知るために、ネットワーク機能毎の関数の分類を行った上で、分類後の関数群をモジュールと定義して、モジュラリティ [14]

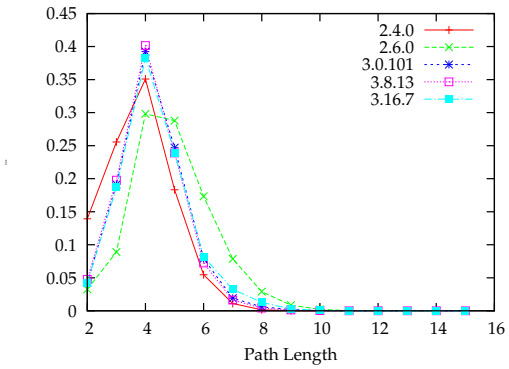



図 6: パス長の分布の推移

表 5: バージョン 3.0 から 3.16 までに増加したネットワーク機能間のリンク数の総増加リンク数に占める割合

| | ip | tcp | udp | sctp | ethernet | socket |
|---|---|---|---|---|---|---|
| ip | 0.038 | 0.002 | 0.001 | 0.006 | 0.001 | 0.024 |
| tcp | 0.009 | 0.039 | 0 | 0 | 0 | 0.029 |
| udp | 0 | 0 | 0.006 | 0 | 0 | 0.010 |
| sctp | 0.013 | 0 | 0 | 0.266 | 0 | 0.064 |
| ethernet | 0.002 | 0 | 0 | 0 | 0.146 | 0.016 |
| socket | 0.016 | 0.004 | 0.002 | 0.005 | 0.004 | 0.130 |

を計算した。また比較として、コールグラフを無向グラフにした上で Louvain 法を用いてモジュール分割を行い、モジュラリティを計算した。図 7 にモジュラリティの推移を示している。図 7 より、ネットワーク機能に基づき分割を行った場合のモジュラリティは Louvain 法により求めたモジュラリティに比べ小さく、ネットワーク機能が独立しておらず、他機能への依存が強いことがわかる。また、モジュラリティの差はバージョンが 2 系列から 3 系列に進むにつれて大きくなっている。これは、Linux カーネルの開発進行に伴い機能構造が複雑になったためと考えられる

いずれのネットワーク機能への依存が強まっているのかを調べるために、各ネットワーク機能へのリンク数の推移を調査した。図 8 に他のネットワーク機能から各ネットワーク機能へのリンク数の推移を示している。図中の各線は他の機能から該当するネットワーク機能へのリンク数を表している。図 8 より、IP や socket のネットワーク機能へのリンク数が増大しており、IP や socket への依存が高まっていることがわかる。

## 5. おわりに

本稿では、Linux カーネルにおけるインターネットプロトコルスタックを用いて、プロトコル間及びプロトコル内におけるネットワーク機能の依存性を分析した。ネットワーク機能が他のプロトコルのネットワーク機能に強く依存しており、この傾向はカーネル開発が進むにつれてわずかではあるが強まることがわかった。これは、NFV や SDN 環境においてネットワーク機能をコンポーネント化するコストが高くなることを意味する。またプロトコル間と同様にプロトコル内においてもネットワーク機能の依存がみられた。実際にネットワーク機能をコン

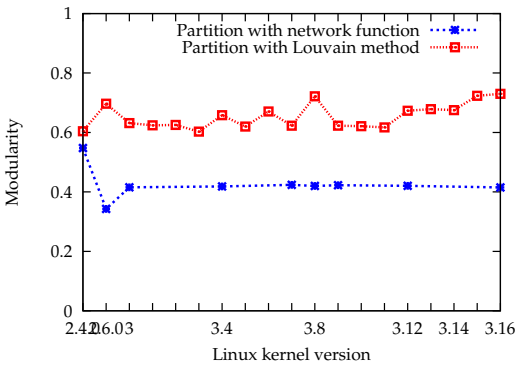



図 7: モジュラリティの推移

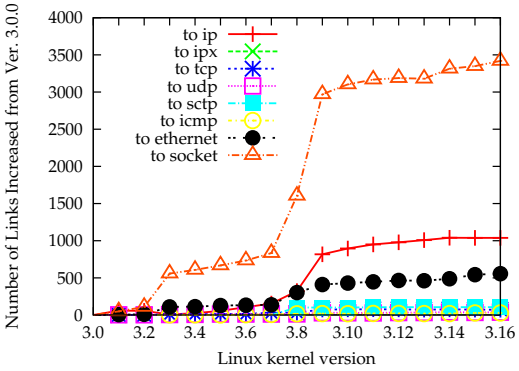



図 8: ネットワーク機能へのリンク数の推移

ポーネント化する場合は、プロトコルが複数のネットワーク機能から構成されるよう行われることが考えられ、この結果からもネットワーク機能のコンポーネント化のコストが高くなると言える。

今回行ったコールグラフに基づく分析はソースコードを用いた静的な分析であり、実際のプログラムの動作を反映したものではない。動的な解析を行う為に Linux カーネルが動作している際の関数呼び出しを分析する予定である。

**謝辞** 本研究の一部は、科学研究費補助金基盤研究 (B)25280029 によっている。ここに記して謝意を表す。

## 文　献

[1] S. Akhshabi and C. Dovrolis, "The evolution of layered protocol stacks leads to an hourglass-shaped architecture," *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 206–217, Aug. 2011.

[2] A. Fischer, J. F. Botero, M. Till Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, pp. 1888–1906, Feb. 2013.

[3] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (SDN): A reference architecture and open APIs," in *Proceedings of ICT Convergence (ICTC), 2012 International Conference on*, IEEE, Oct. 2012.

[4] B. Partha, Z. Shuqiang, C. Pulak, L. Sang-Soo, L. J. Hyun, and M. Biswanath, "Software-defined optical networks (SDONs): A survey," *Photonic Network Communications*, vol. 28, pp. 4–18, Aug. 2014.

[5] J. Batalle, J. Ferrer Riera, E. Escalona, and J. Garcia-Espin, "On the implementation of NFV over an OpenFlow infrastructure: Routing function virtualization," in *Proceedings of IEEE SDN for Future Networks and Services (SDN4FNS)*, pp. 1–6, Nov. 2013.

[6] L. Battula, "Network security function virtualization (NSFV) towards cloud computing with NFV over Openflow infrastructure: Challenges and novel approaches," in *Proceedings of International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1622–1628, Sept. 2014.

[7] "The Linux Kernel Archives." Available at: `http://www.kernel.org`. Accessed: 1 Feb. 2015.

[8] M. Gorman, "Codeviz: A callgraph visualiser." Available at: `http://www.csn.ul.ie/~mel/projects/codeviz/`. Accessed: 1 Feb. 2015.

[9] M. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, pp. 323–351, 2 2005.

[10] K.-K. Yan, G. Fang, N. Bhardwaj, R. P. Alexandera, and M. Gersteinb, "Comparing genomes to computer operating systems in terms of the topology and evolution of their regulatory control networks," *Proceedings of the National Academy of Sciences*, vol. 107, pp. 9186–9191, May 2010.

[11] 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.

[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics*, vol. 2008, pp. 10008–10019, Oct. 2008.

[13] T. Dreibholz, E. P. Rathgeb, I. Rüngeler, R. Seggelmann, M. Tüxen, and R. R. Stewart, "Stream control transmission protocol: Past, current, and future standardization activities," *IEEE Communications Magazine*, vol. 49, pp. 82–88, Apr. 2011.

[14] K. A. Eriksen, I. Simonsen, S. Maslov, and K. Sneppen, "Modularity and Extreme Edges of the Internet," *Physical Review Letters*, vol. 90, pp. 1–4, Apr. 2003.