特別研究報告

題目

インターネットプロトコルスタックの進化過程の評価を目的とした Linux カーネルの分析

指導教員

村田 正幸 教授

報告者

宮川裕考

2015 年 2 月 13 日

大阪大学 基礎工学部 情報科学科

平成 26 年度 特別研究報告

インターネットプロトコルスタックの進化過程の評価を目的とした Linux カーネルの分析

宮川裕考

内容梗概

インターネットでは IP（Internet Protocol）を中心とするプロトコルスイートが用意されており、インターネットのプロトコルスタックを利用した様々な通信サービスが提供されている。現状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 (Transmission Control Protocol) / UDP (User Datagram Protocol) が固定的に利用されている。しかし、スマートフォンやタブレット端末が普及を背景に、インターネットを介して提供されるサービスは多様化し、これまでになかった新たなサービスへの要求が高まりつつある。ところが、インターネットを用いるためには IP を使用せざるを得ず、IP が提供できない機能を要する新たなサービスの展開が阻害される状況にあり、プロトコルスタックの硬直化の弊害が顕在化している。最近は、ネットワーク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って柔軟なサービス提供を可能とする NFV(Network Function Virtualization) や SDN (Software Defined Networking) などの方策も検討されつつある。しかし、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っては、プロトコル間の機能分担やプロトコルを構成する機能群の機能分担が適切になされていることが重要となる。これは、適切な機能分担がなされていない場合に多数のプロトコルに改変を加えなければならないためである。

本報告では、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機能結合とその進化の様相を明らかにする。ここでいう様相とは、ネットワーク機能がどのように構成され、また、カーネル開発とともに機能間の接続性の変遷である。Linux カーネルのバージョン 2.6.27 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、ノード数・リンク数は増加しているものの、次数分布やパス長の分布は変化していないことが明らかとなった。しかし、機能間の独立性を示す尺度であるモジュラリティを評価した結果、機能間の独立性が高くないことがわかった。

主な用語

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

目次

図目次

表目次

1 はじめに

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。それに伴い BitTorrent や Skype といった新たなアプリケーションプロトコルが登場し、利用されつつある。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は置き換わることなく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、他のプロトコルが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。しかしその一方で、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。また固定化した中間層のプロトコルに依存するプロトコルが多いため、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。

インターネットのプロトコルスタックは開発当初から砂時計型であった訳ではない。例えば 1990 年代初頭までは IPv4 に対抗するプロトコルとして、IPX や X.25 のネットワーク層プロトコルが使用されていた。しかしインターネットを取り巻く環境が変化に対応するため、プロトコルスタックは進化していく必要があった。その過程において IPv4 が主に使用されるようになったのである。つまり、砂時計型のプロトコルスタックは進化の結果として生じたものなのである。現在のインターネットで見られる砂時計型のプロトコルスタックがどのような過程で現れたかをモデル化した研究として文献 [1] がある。文献 [1] ではプロトコルスタックの進化モデルを導入し、プロトコルスタックが砂時計型となる要件を明らかにしている。プロトコルスタックを進化可能性の高いものとするための方策として、使用されていないプロトコルでも保守し続けること、同じ層で使用されている他のプロトコルと機能競合しないプロトコルを導入すること、汎用性を有するプロトコルが砂時計でいうくびれにあたる中間層に位置するようプロトコルスタックを再設計すること、が挙げられている。しかしながら、現在のインターネットで用いられているプロトコルスタックでは、もはや新しいサービス要求の高まりに対し、自身のプロトコルスタックを改変していくことは本質的に困難である [2]。柔軟かつ迅速なネットワーク構築の要求への対処方法として、最近では

このような既存のプロトコルの制約から逃れる為に、全く新しいプロトコルスタックを採用した次世代のインターネットアーキテクチャや、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 [3–8]。

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、プロトコル間の機能分担が適切になされていることが重要となる。これは、仮に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ためである。インターネットのプロトコルスタックは、原則として OSI 参照モデルに基いて各プロトコルで機能の分担が行われていると言われているものの、プロトコルスタックの実装によっては適切な機能分担が行われていないことも指摘されている。従って、インターネットを構成するプロトコルがどのようなスタック構成をなしているか、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。また、プロトコル間の機能分担だけではなく、プロトコル内の機能分担、すなわち、あるプロトコルを構成する機能群の構成についても、ネットワーク機能のコンポーネント化にあたって適切になされているかを理解することも重要となる。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。

本報告では、Linux カーネル [9] 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。Linux カーネルにはインターネットで利用される多数のプロトコルが実装されており、また Linux カーネルの現在から過去に至るまでのバージョンはすべて公開されているため、調査を行うのが容易である。Linux カーネルの進化を分析した研究として、文献 [10–12] 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。しかし、文献 [10] ではプロトコルスタックではなく Linux カーネル全体を分析対象としている。文献 [11] では Linux カーネルからコールグラフを作成し、その進化を次数分布や平均パス長などの指標を用いて分析しているが、これも Linux カーネル全体を対象としたものである。しかし、ネットワーク機能間及び機能内の結合を理解するには、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。

そこで本報告では、Linux カーネルのコールグラフに基づき、プロトコル間およびプロトコル内

におけるネットワーク機能同士の依存関係を明らかにし、またそれが開発進行によりどのように変化していったかを明らかにする。

2　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

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。本報告では、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。図 1 に関数 kernel_sendmsg のプログラムコードと生成されるコールグラフの例を示す。

```
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; }
```

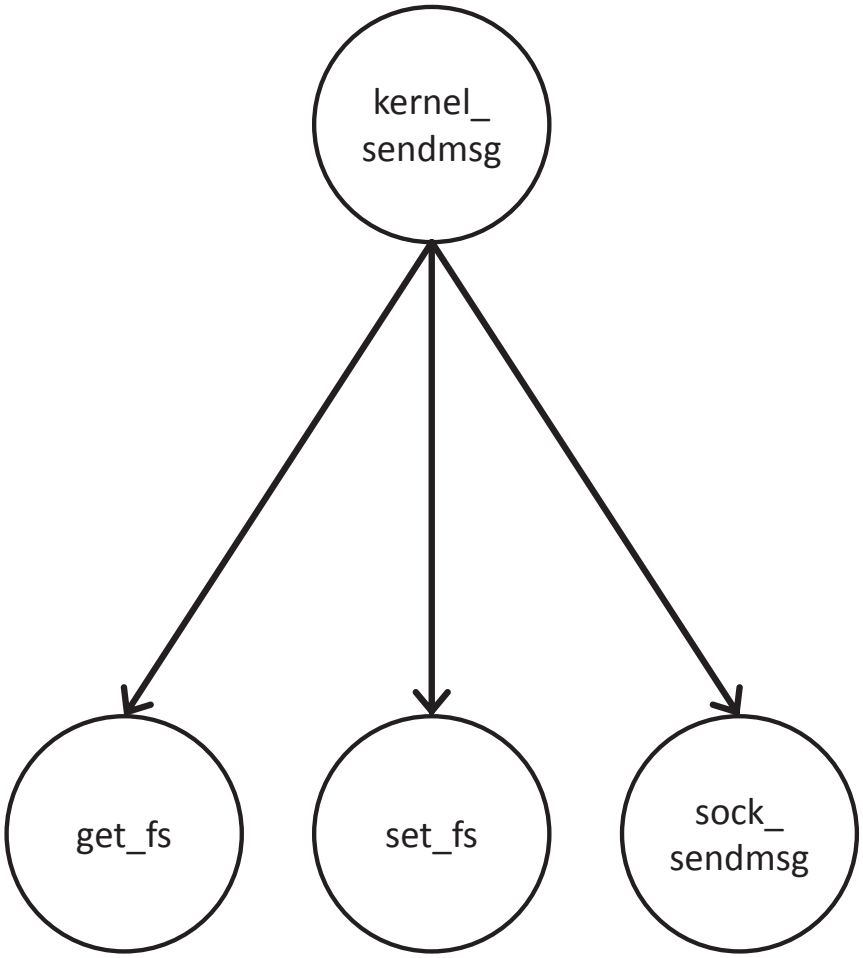

図 1　関数 kernel_sendmsg のプログラムコードと生成されるコールグラフ

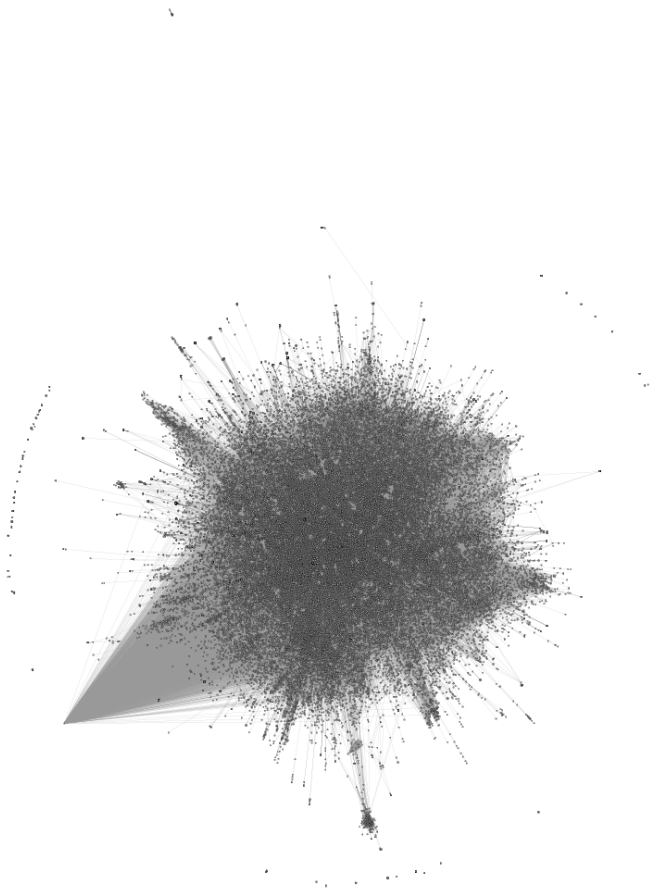

図 2　Linux カーネル 2.6.27 全体のコールグラフ

2.2　Linux カーネルのコールグラフの分析

コールグラフは CodeViz [13] 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。

先述の手順により作成したコールグラフを図 2 に示す。対象とした Linux カーネルのバージョンは 2.6.27 である。このグラフのノード数は 71,140 で、リンク数は 272,501 である。このコールグラフがの次数分布は図 3 に示すとおりである。分布はべき則に従っている。すなわち、一部

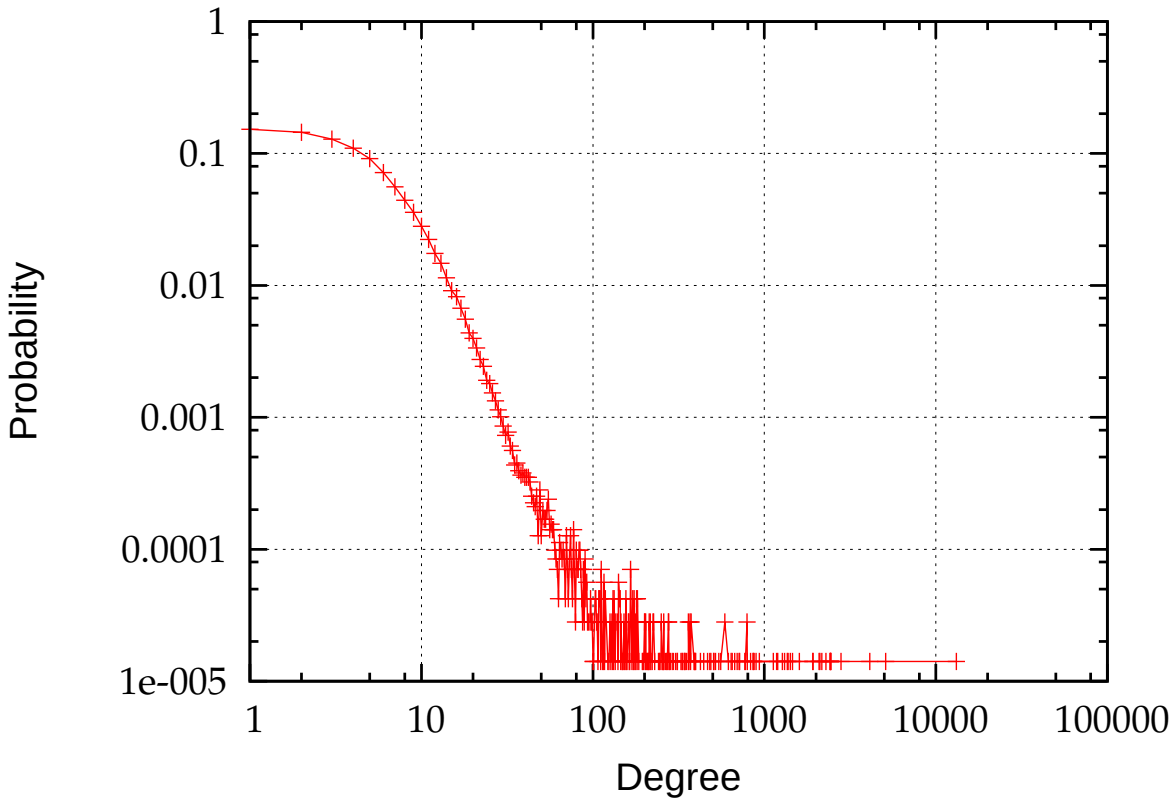

図 3　Linux カーネル 2.6.27 のコールグラフの次数分布

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という性質を有している [14–18]。Linux カーネル全体のコールグラフにおいて、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となっている。

表 1　Linux カーネルにおけるハブノードの例

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
netdev_priv	3474
__dynamic_pr_debug	3323
kzalloc	3213
dev_err	2969
constant_test_bit	2941
writel	2752
readl	2679
IS_ERR	2546
warn_slowpath_null	2432
kmalloc	2318

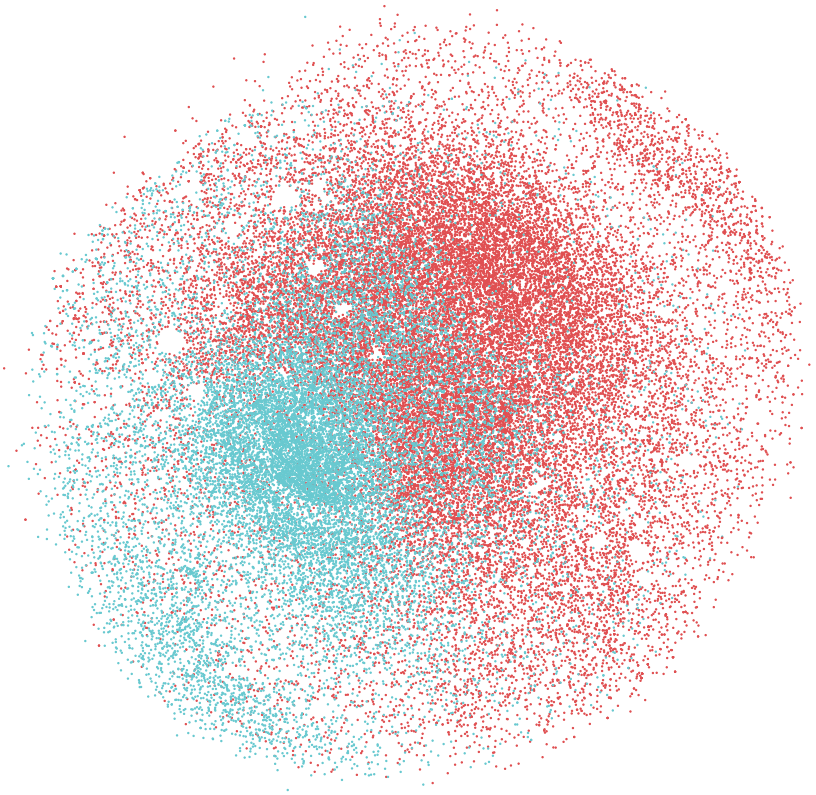

図 4　Linux カーネル 3.16.7 全体のコールグラフ

2.3　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の分析

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、本報告ではネットワーク機能の接続構造に着目しているので、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。図 4 に示す Linux カーネル全体のコールグラフに、上記の方法で抽出したネットワーク機能に関連した関数を青色ノードで、それ以外は赤色ノードで示している。ネットワーク機能関連の関数のみからなるグラフのノード数は 11,545 であり、リンク数は 41,320 である。

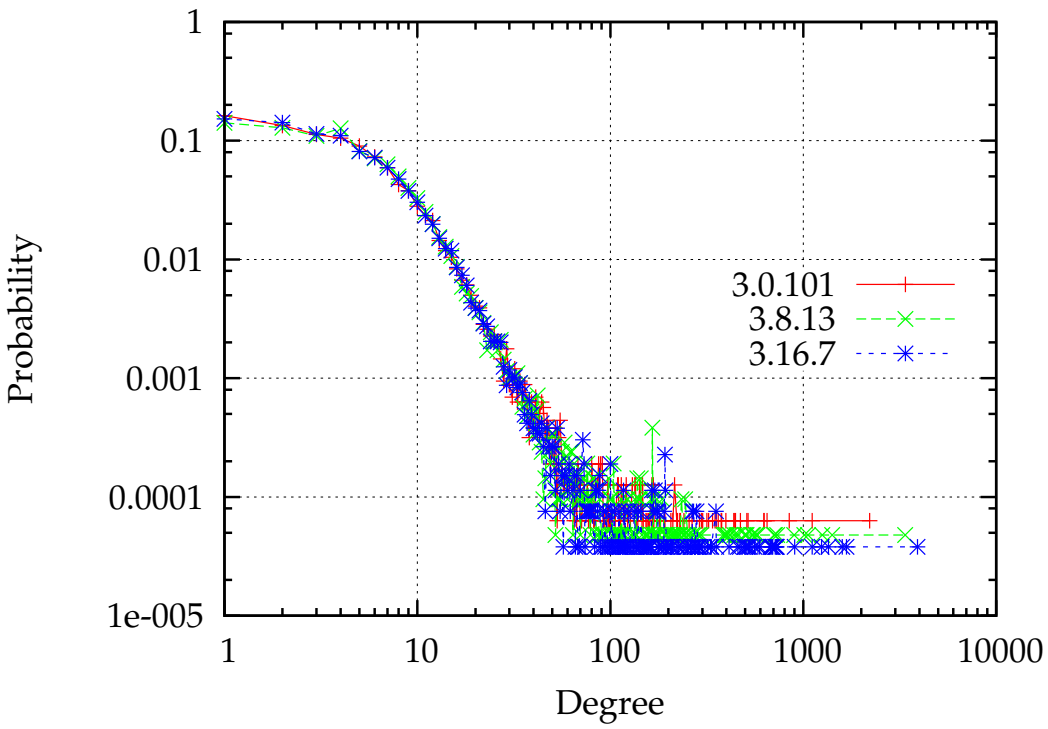

図 5　ディレクトリ net 配下の関数群からなるコールグラフの次数分布

図 5 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。

表 2　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

2.4　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。

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 2 に示す。

表 3　Linux カーネル 3.16.7 におけるグループ間のリンク数

	ip	ipx	tcp	udp	sctp	icmp	ethernet	socket	nla	dst	nlmsg	netfilter	xfrm	router	others
ip	2053	0	29	14	1	77	7	1043	247	210	95	150	26	178	1164
ipx	0	140	0	0	0	0	0	91	0	0	0	0	0	0	34
tcp	450	0	1251	0	0	2	0	872	10	73	0	39	0	7	308
udp	144	0	12	157	0	6	0	300	0	15	0	19	0	1	121
sctp	18	0	0	0	14	0	0	16	0	0	0	0	0	0	1
icmp	154	0	0	0	0	101	1	156	8	28	0	38	6	14	155
ethernet	30	0	0	0	0	0	1927	470	200	2	160	7	0	50	978
socket	165	6	100	17	0	3	26	1337	2	75	2	10	14	4	539
nla	0	0	0	0	0	0	0	2	22	0	1	0	0	0	1
dst	26	0	0	0	0	0	1	43	0	99	0	4	16	6	59
nlmsg	1	0	0	0	0	0	0	9	15	0	29	0	0	4	7
netfilter	65	0	7	0	0	1	2	77	5	14	0	440	2	7	250
xfrm	7	0	0	0	0	0	0	69	24	58	57	4	469	2	259
router	79	0	0	0	0	1	1	59	64	55	63	1	1	152	204
others	1038	1	58	22	0	40	304	3180	815	229	316	473	218	342	15804

2.5　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 3 に示す。表 3 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。

リンク数だけを見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 [19] によりモジュール分割を行った際の値約 0.73 に比べて低いので、一つのネットワーク機能は他のネットワーク機能の関数にも依存していると言える。

3　インターネットプロトコルスタックの変遷分析

2011 年 7 月にリリースされたバージョン 3.0 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。

3.1　コールグラフの構造的特徴

図 6 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 6 より、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 Stream Control Transmission Protocol (SCTP) [20] の導入が進んだためと考えられる。図 7 は kernel 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しているためである。図 7 より、バージョン間の次数分布に大きな変化はないことがわかる。図 8 は kernel 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 8 よりバージョン間にパス長に大きな変化はなく、構造的特徴が開発が進んでも維持されていることがわかる。

3.2　ネットワーク機能間の接続構造の変遷

図 9 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を示している。左側のネットワーク機能から下側のネットワーク機能へのリンク数の増加が顕著な場合、交差する区間の色が赤に近いものとなる。図 9 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 [21] 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

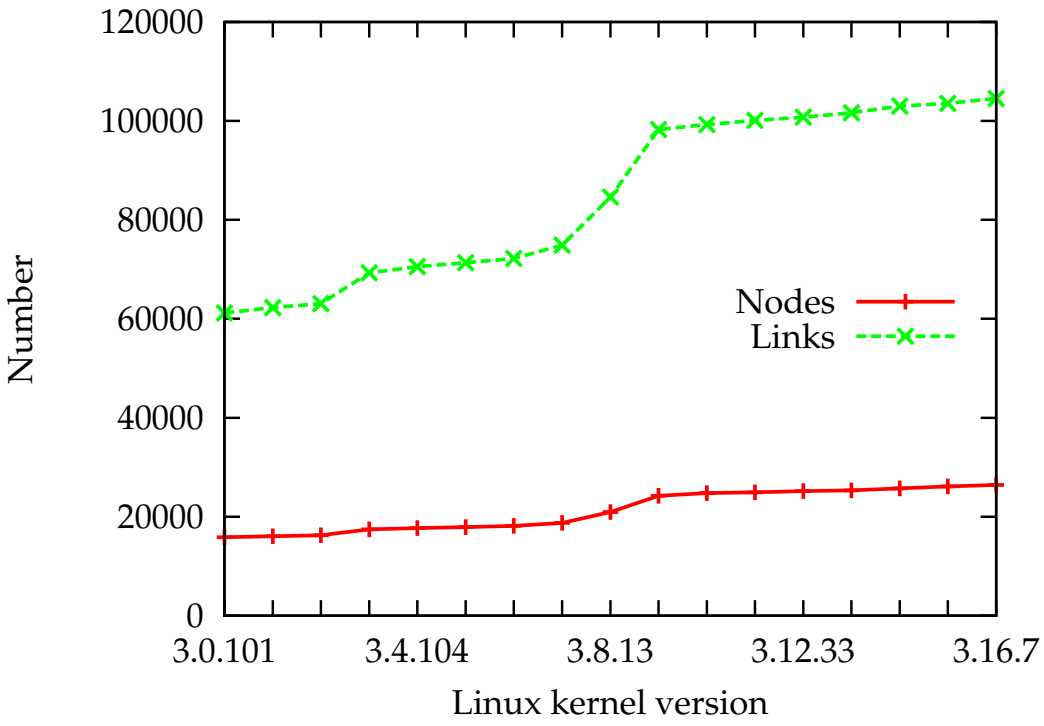

図 6　ノード数およびリンク数の推移

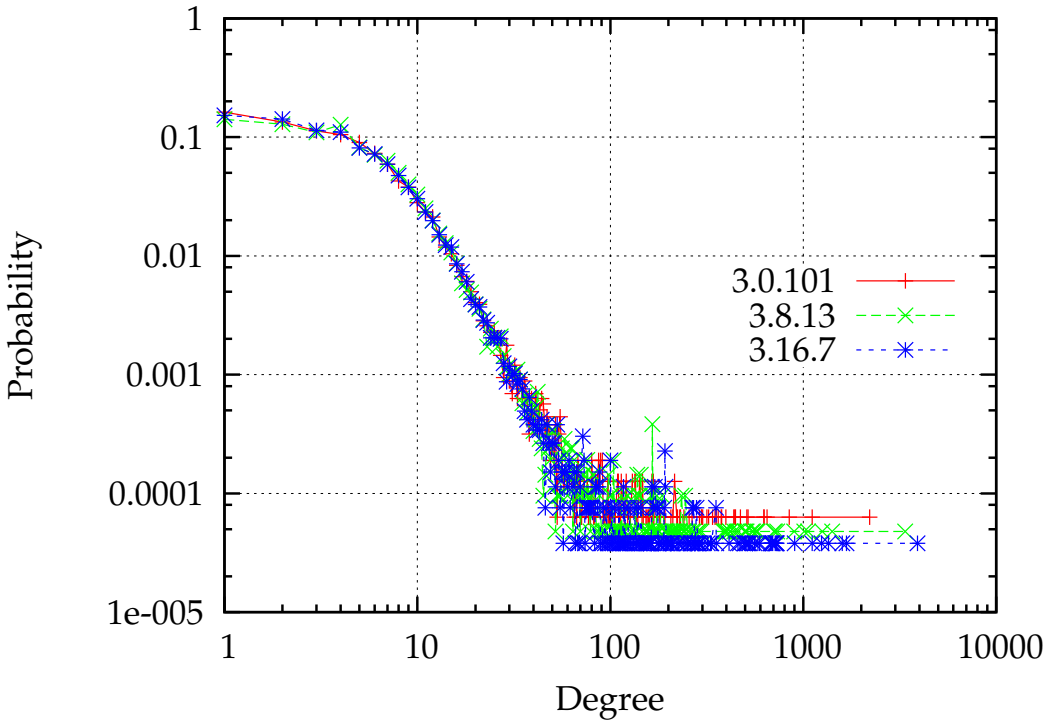

図 7　次数分布の推移

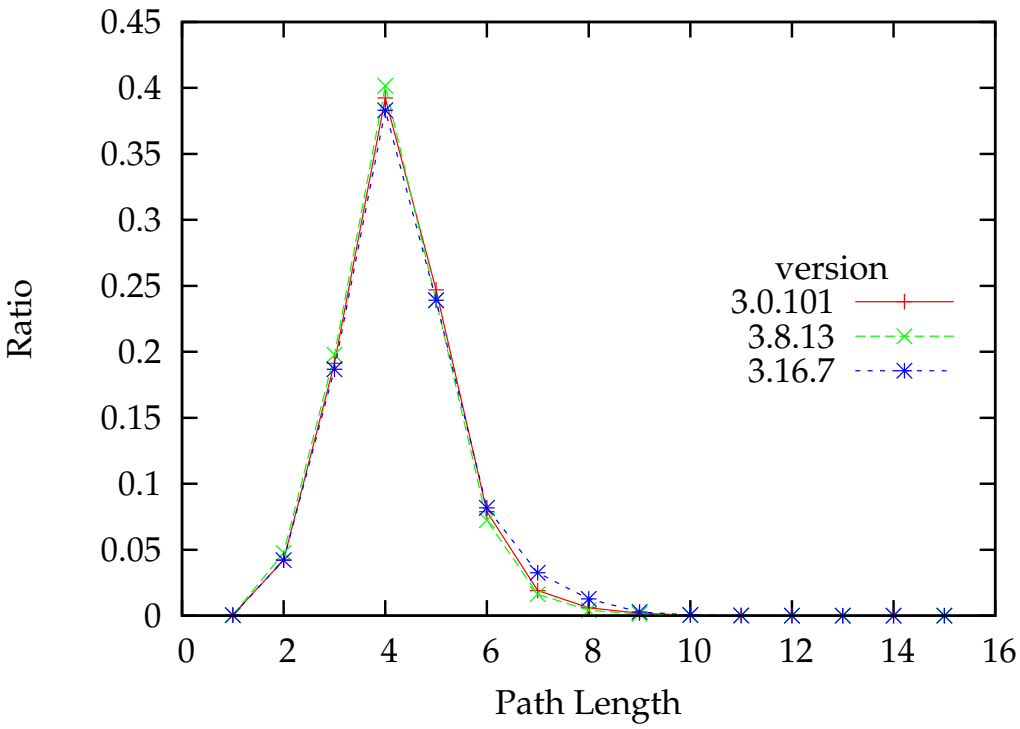

図 8　パス長の分布の推移

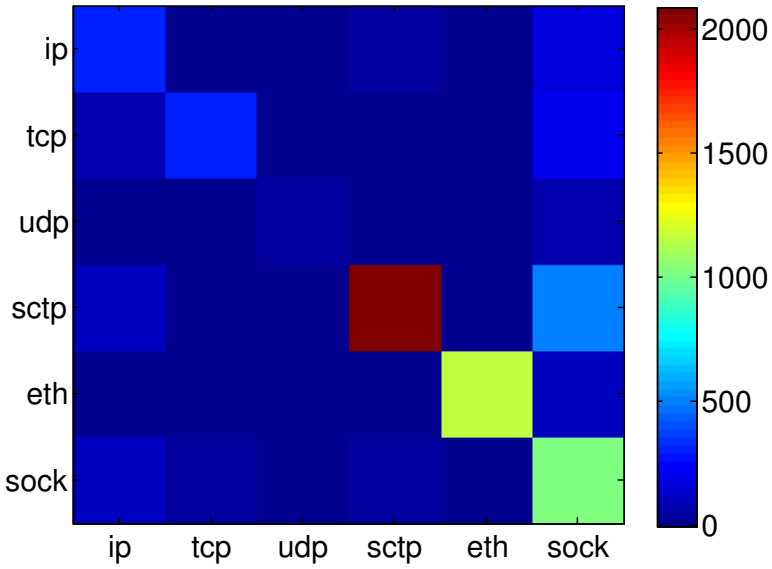

図 9　バージョン 3.0 から 3.16 までに増加したネットワーク機能間のリンク数

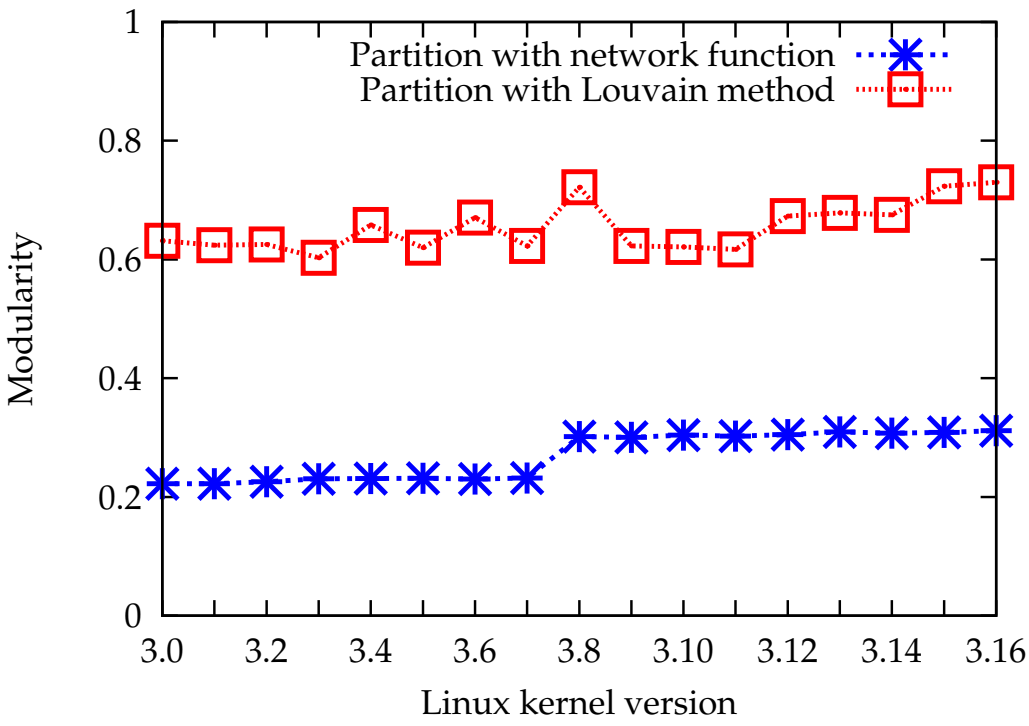

図 10　モジュラリティの推移

行い、モジュラリティを計算した。図 10 にモジュラリティの推移を示している。図 10 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。

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 11 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 11 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。

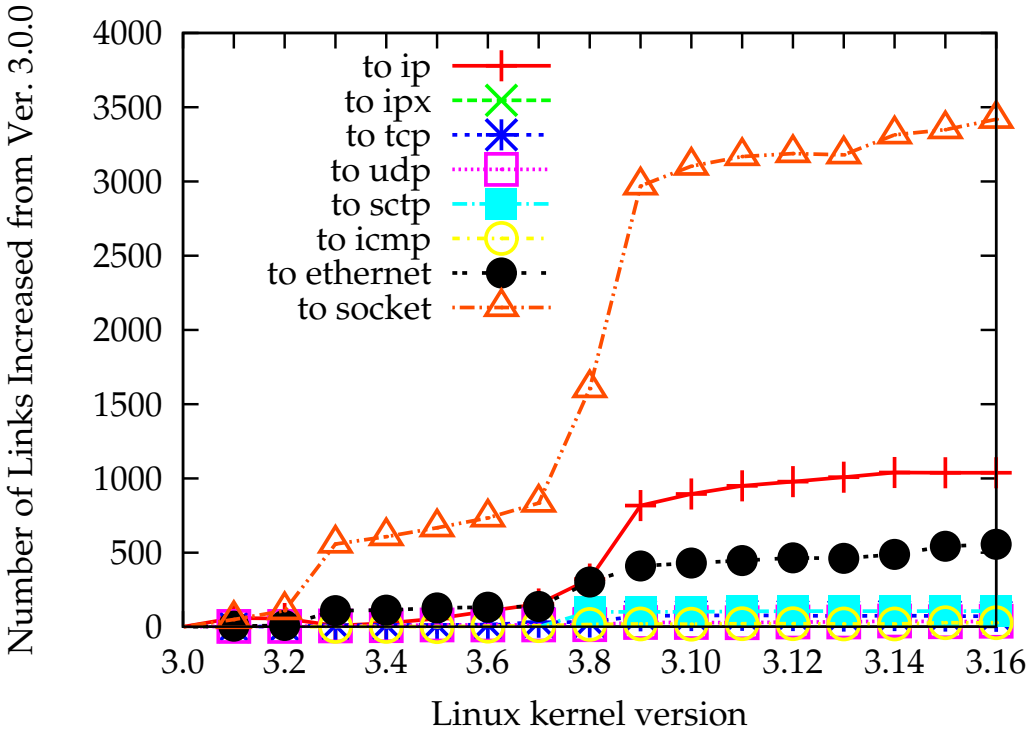

図 11　ネットワーク機能へのリンク数の推移

4 おわりに

本報告では、ネットワーク機能をコンポーネント化するにあたって課題となる、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事例として、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明らかにした。プロトコル間においてネットワーク機能の依存がみられた。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、ネットワーク機能をコンポーネント化するコストが高いことがわかった。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いことが示された。

今回行ったコールグラフに基づく分析は静的な分析であり、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

本報告を終えるにあたり、熱心に御教授、御指導下さいました大阪大学大学院情報科学研究科の村田正幸教授に厚く御礼申し上げます。ならびに、平素より丁寧に直接御指導下さいました大阪大学大学院情報科学研究科の荒川伸一准教授に心より御礼申し上げます。加えて、本報告を作成するにあたり、適切な助言を下さいました大阪大学大学院情報科学研究科の大下裕一助教、小泉佑揮助教そして大阪大学大学院経済学研究科の小南大智助教に感謝いたします。最後に、日頃より様々なご助言とご助力を頂きました、大阪大学大学院情報科学研究科の中田侑氏、シン・ルー氏、大場斗士彦氏、四條能伸氏、井上昴輝氏をはじめとする村田研究室の皆様にもお礼申し上げます。

参考文献

[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] AKARI プロジェクト, **新世代ネットワーク・アーキテクチャ** *AKARI* **概念設計書改訂版** *(ver2.0)*. 2009.

[3] J. Pan, S. Paul, and R. Jain, "A survey of the research on future Internet architectures," *IEEE Communications Magazine*, vol. 49, pp. 26–36, July 2011.

[4] 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.

[5] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (SDN): A reference architecture and open APIs," in *Proceedings of IEEE International Conference on ICT Convergence*, pp. 360–361, Oct. 2012.

[6] 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.

[7] 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.

[8] 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.

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

[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] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Transactions on Software Engineering*, vol. 33, pp. 687–708, Oct 2007.

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

[14] T. Bu and D. Towsley, "On distinguishing between Internet power law topology generators," in *Proceedings of IEEE INFOCOM*, pp. 638–647, June 2002.

[15] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. J. Shenker, and W. Willinger, "The origin of power laws in Internet topologies revisited," in *Proceedings of IEEE INFOCOM*, pp. 608–617, June 2002.

[16] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," *ACM SIGCOMM Computer Communication Review*, vol. 29, pp. 251–262, Aug. 1999.

[17] C. Gkantsidis, M. Mihail, and A. Saberi, "Conductance and congestion in power law graphs," *SIGMETRICS Performance Evaluation Review*, vol. 31, pp. 148–159, June 2003.

[18] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topologies, power laws, and hierarchy," *Computer Communication Review*, vol. 32, pp. 1–26, Jan. 2002.

[19] 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.

[20] 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.

[21] 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.