# On the Structural Evolution of Linux Kernel

Ganesh Narayan, K Gopinath
Department of Computer Science and Automation,
Indian Institute of Science, Bangalore

***Abstract*— Software systems, subjected to exogenous and endogenous requirements, grow and evolve to become complex and intricate, consequently difficult to maintain. Understanding how software evolves is critical to maintain the software. In this work we study software evolution in terms of their induced graph evolution. Through a case study, we explore the dynamics of such a evolution and show how it helps us comprehend the process of software evolution.**

## I. Introduction

LARGE software systems must evolve over time, or they risk loosing community following. Evolution of a software is brought forth by: fixing defects, adding new features, tuning the system, supporting new hardware/platform, changing the design and a wide range of other requirements. In adopting to this ever changing software landscape, software systems – not unlike biological systems – evolve to become "better", generally in terms of features. However, as a software ages and grows it becomes increasingly difficult to comprehend and maintain. In order to maintain a software better, it is imperative that we study and understand this process of software evolution. In this work we propose a means of looking at software evolution that, we hope, would improve our comprehension.

A software is defined by a set of modules and their interactions. Taken this way, software naturally gives raise to a graph structure with nodes being modules and edges depicting the interactions. One can now see that the process of software evolution can now equivalently be formulated as the evolution of these graphs;

In this work we study the evolution of a software system – Linux kernel, by studying its induced graph structure – with functions as the nodes. Linux is a free, Unix-like kernel created by Linus Torvalds and is subsequently developed by developers around the world. It was originally developed for x86, but has since been ported to numerous other platforms. We chose Linux kernel becasue:

- The source code for Linux is freely available.
- Linux is one of the oldest and most popular opensource system and is under constant development. This helps us to see how a software reacts to realistic requirements.
- Linux supports a vast number of hardware platforms, has a large set of funtionalities to offer and is developed by hundreds of individuals. Any invariant observed over such a disparate system is likely to be universal.
- Linux has evolved from being a pet project to a mainstream operating systems; it should guide us to the large scale evolution of software systems.
- Operating system kernels are one of the hardest systems to develop and debug; so maintenance cycle of Linux should help us understand the small scale evolution.

| * | **2.2.12** | **2.4.16** | **2.4.18** | **2.4.20** | **2.6.8** | **2.6.12** | **2.6.13** |
|---|---|---|---|---|---|---|---|
| **2.2.12** | 1.0000 | 0.5846 | 0.5710 | 0.5444 | 0.0895 | 0.0853 | -0.1280 |
| **2.4.16** | 0.5846 | 1.0000 | 0.9641 | 0.9058 | 0.1441 | 0.1398 | -0.0528 |
| **2.4.18** | 0.5710 | 0.9641 | 1.0000 | 0.9189 | 0.1413 | 0.1366 | -0.0652 |
| **2.4.20** | 0.5444 | 0.9058 | 0.9189 | 1.0000 | 0.1460 | 0.1452 | -0.0477 |
| **2.6.8** | 0.0895 | 0.1441 | 0.1413 | 0.1460 | 1.0000 | 0.5663 | -0.2728 |
| **2.6.12** | 0.0853 | 0.1398 | 0.1366 | 0.1452 | 0.5663 | 1.0000 | 0.0056 |
| **2.6.13** | -0.1280 | -0.0528 | -0.0652 | -0.0477 | -0.2728 | 0.0056 | 1.0000 |

TABLE I
Function correlation matrix

| * | **2.2.12** | **2.4.16** | **2.4.18** | **2.4.20** | **2.6.8** | **2.6.12** | **2.6.13** |
|---|---|---|---|---|---|---|---|
| **2.2.12** | 1.0000 | 0.4689 | 0.4068 | 0.3897 | 0.1779 | 0.1765 | 0.1087 |
| **2.4.16** | 0.8409 | 1.0000 | 0.8383 | 0.8033 | 0.2943 | 0.2928 | 0.1980 |
| **2.4.18** | 0.8584 | 0.8774 | 1.0000 | 0.9514 | 0.2637 | 0.2613 | 0.1770 |
| **2.4.20** | 0.8542 | 0.8728 | 0.9939 | 1.0000 | 0.2756 | 0.2746 | 0.1863 |
| **2.6.8** | 0.6060 | 0.5711 | 0.6462 | 0.6480 | 1.0000 | 0.7359 | 0.3134 |
| **2.6.12** | 0.584 | 0.5533 | 0.6320 | 0.6337 | 0.8367 | 1.0000 | 0.4255 |
| **2.6.13** | 0.7350 | 0.6801 | 0.7542 | 0.7572 | 0.8526 | 0.876 | 1.0000 |

TABLE II
$|I \backslash O|$ degree correlation matrix

With functions as nodes, edges depict the caller-callee relation between functions; the graph structure thus becomes the static call graph or simply, call graph, of Linux Kernel. This graph is simple, sparse, directed and possibly cyclic. Given a static call graph, indegree depicts the number of functions that call the particular function and outdegree depicts the number of functions the particular function calls.

## II. Methodology & Results

We use the CodeViz call graph extractor to extract call graphs from Linux sources. We studied the static call graphs of seven Linux kernel versions ranging from 2.2 to 2.6, spanning nearly 6 years.

### A. Nodes & Edges

Figure I shows how the kernel size, in terms of number of nodes and edges, evolved over time. Contrary to the intuition – that huge softwares are complex and difficult to understand, hence to modify, addition rate of new functions and edges increases as the software becomes larger and complex. In a sense, software appears to evolve like a gravititating body: bigger the software, better is the rate of change.

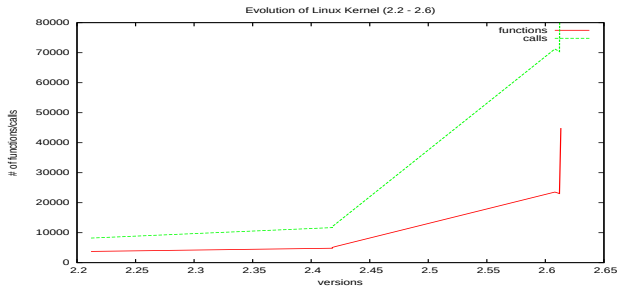


Fig. 1. Program Size Vs. Version

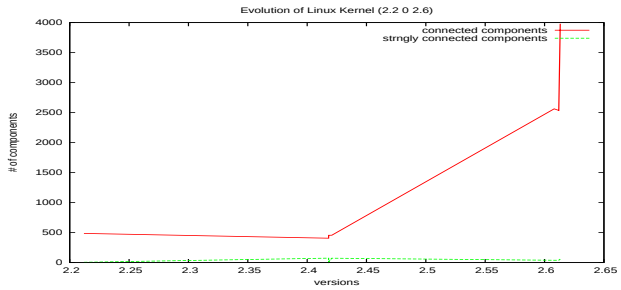


Fig. 2. Number of Components Vs. Version

Figure II shows how number of connected and strongly components grow. It appears that the number of weakly connected components increase at a rate comparable to the rate at which new edges are added. Also, there is a small but an *increasing* number of strongly connected components, hinting at increased interactions through mutual recursion.

Table III shows the percentage of nodes in the largest connected and strongly connected component, respectively. The fraction of nodes that constitute the connected component remain the same despite the significant increase in the graph size. The authors believe that it is due to increased interaction between layers – a result of aggressive crosslayer optimizations. However, though the gigantic connected component absorbs nearly 80-90% of nodes, the remaining nodes themselves are fragmented into a large number of connected components. The fraction of nodes that constitute the strongly connected components steadily decreases as time progresses: either the functions tend to get "fine granular" – as the versions grow – so that pseudo mutual recursion present in earlier versions disappear; or, mutual recursion is consciously penalized as the system evolves.

### B. Node Correlation

In this section we study how function names are correlated in various versions of Linux. We computed the total number of functions $N$ across all versions and each version is represented as a bit vector $F$ of size $N$. if function $i$ is present in a version, $F[i]$ is set to 1; reset otherwise.

Having arrived at these seven vectors we calculate the $7 \times 7$ correlation matrix $C$ that compares each vector with every other vector; $C[i][j]$ depicts the similarity between $i^{th}$ and $j^{th}$ vectors/versions. To compute pairwise similarity, we use Pearson's Coefficient $\rho$ defined as

| $*$ | **2.2.12** | **2.4.16** | **2.4.18** | **2.4.20** | **2.6.8** | **2.6.12** | **2.6.13** |
|---|---|---|---|---|---|---|---|
| **Connect %N** | 84 | 89 | 89 | 89 | 88 | 88 | 90 |
| **SConnect %N** | 14 | 7 | 4.9 | 2.2 | 0.8 | 0.9 | 0.6 |

TABLE III
% OF NODES IN THE LARGEST CONNECTED/STRONGLY CONNECTED COMPONENT

$$\rho = \frac{\sum(x-\bar{x})(y-\bar{y})/N}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2/N^2}}$$

Table I depicts the results. Please note that the values are symmetric about the main diagonal as correlation between ($F_i$, $F_j$) is same as ($F_j$, $F_i$).

$+1$ in a cell implies perfect positive correlation; 0 represents no correlation; $-1$ implies perfect negative correlation. As shown in the table, versions that are chronologically closer exhibit strong positive correlation and share similar correlation profiles when compared against other versions. As we move away from the main diagonal, $\rho$ decreases, implying significant changes even in once-moved versions. Version 2.6.13 shares weakest correlation with rest of the system. This, we believe, is because 2.6.13 is the latest and also the only "odd", hence unstable release.

### C. Degree Correlations

Next we would like to compute the correlation between degrees across versions. For this purpose we extract two vectors $I$ and $O$ from *each* version, where $i^{th}$ element contains in and out degrees respectively. We then compute two $7 \times 7$ correlation matrices, $C_i$ and $C_o$ for $I$'s and $O$'s respectively. Since both $C_i$ and $C_o$ symmetric, we could represent the result as matrix $C$ whose lower triangular component is $C_i$ and upper triangular component is $C_o$. Table II shows this matrix. First thing to notice is that all entries are positive and are significantly larger than zero. Thus there exists a non trivial positive degree correlation between degrees of *any* two versions of the software; thus, it appears that the relative in/out degree structure of the software tend to remains similar. The correlation matrix is highly skewed about the main diagonal, suggesting that the indegree and outdegree evolve differently. It is also to be noted that the indegree correlations between any two versions is far more pronounced compared to their corresponding outdegree correlation: indegree and outdegree distributions do not appear to evolve similarly.

## III. Conclusion

There is lot to be analyzed and understood but we hope we gave a flavour of what it means to study software evolution in terms of their corresponding graph structure evolution.

We intend to study: how different subsystems of Linux kernel evolve over time; what is the relative "rate" of evolution in these subsystems; if bugs and evolution rate are positively correlated. We also plan to study the algebraic/topological characteristics of these graphs to understand software evolution as a dynamical system.