Evolution of Apache Open Source Software

Haoran Wen, Raissa M. D'Souza, Zachary M. Saul, and Vladimir Filkov

University of California, Davis CA 95616, USA;
hrwen@ucdavis.edu, rmdsouza@ucdavis.edu, zmsaul@ucdavis.edu,
vfilkov@ucdavis.edu

1 Software: A General Paradigm for Network Systems?

Our modern infrastructure relies increasingly on computation and computers. Accompanying this is a rise in the prevalence and complexity of computer programs. Current software systems (composed of an interacting collection of programs, functions, classes, etc.) implement a tremendous range of functionality, from simple mathematical operations to intricate control systems. Software systems are inherently extendable and tend to gain new functionality over time. Modern computers and programming languages are Turing complete and, thus, capable of implementing any computable function no matter how complex. The interdependencies between the elements of a software system form a network, and, therefore, we believe software systems can provide useful prototypic examples of how to build complex networked systems which require minimal maintenance, are robust bugs to and yet are readily extendable. Thus we ask: What makes for good design in software systems?

We are particularly interested in open source software (OSS)—software with source code that is freely available for download and modification. A typical OSS project is a collaborative effort by volunteers, with no central authority assigning development tasks. Instead individuals, or self-organized teams of developers, fix bugs and maintain and extend the code. In OSS, modularity is essential [1, 2], and remarkably, the software resulting from an OSS process can rival or even surpass the quality of commercial software [3, 4].

Software systems are always evolving, responding to user demands for "bug fixes" and new features. Invariably, systems grow in size and complexity, eventually becoming difficult to parse, maintain and extend further. In response to this, developers refactor their systems [5], streamlining and restructuring the entire code base. Thus there are several strong analogies between OSS systems and biological systems. Both classes of systems are inherently modular, readily evolvable, must be robust to anomalies and experience periods of punctuated equilibrium [6]. Yet high-confidence data on the structure of OSS, unlike data on biological networks, is easily obtained for minimal cost.

N. Ganguly et al. (eds.), *Dynamics On and Of Complex Networks*,
Modeling and Simulation in Science, Engineering and Technology,
DOI: 10.1007/978-0-8176-4751-3_12,

We analyze a series of fifty monthly snapshots of the function call graph of the Apache 2.0 HTTP Server (called Apache herein). Apache is the most popular web server on the Internet, and has been since 1996 [7]. It is a mature, well-established OSS project managed by a group of volunteers world-wide; to date, hundreds of users have contributed to the code base. Apache is written in the C programming language, a procedural language. The basic elements are functions that explicitly invoke one another through function calls which express the command flow of the program/system. In object oriented systems, in contrast, the software networks are made of edges representing abstract relationships between objects, such as inherits, invokes, etc.

Motivated by advances in network science, we first analyze a collection of measures on global properties of the Apache call graphs. Certain measures behave consistently and we quantify their baselines. Moreover, we find that punctuated changes in these global measures can signal when a more detailed, fine-grained examination of code structure is required. Jumps in global properties can indicate major refactorings, but can also result from restructuring just a few functions (and radically reduce interdependencies). We then turn our focus to a bottom-up approach, studying how observable attributes of the Apache call graph interact using exponential random graph models. Ultimately, by coupling top-down and bottom-up approaches, we want to extract how code is restructured over time to achieve better design. As a mature project, Apache is more in "maintenance" than in growth mode, and the details of changes can be subtle. Yet, these changes may be especially important given that a major expense associated with software is maintenance [8].

Interest in OSS spans multiple communities, from software engineering, to network science, to economics and organizational behavior. Raymond's seminal work [1] is an excellent review of the latter, contrasting the "cathedral" organization of proprietary software to the open "bazaar" nature of open source. Perhaps the first work to consider software systems as complex networks was by Valverde, Ferrer Cancho and Solé [9], in which they show software collaboration graphs have "scale-free" properties which may result from optimal design. Shortly thereafter, Myers conducted a detailed investigation of software collaboration graphs [10], quantifying many features we discuss herein. Both [9] and [10] focus primarily on *object oriented* software (unlike Apache, which is procedural software), looking at one time snapshot of the collaboration network between classes and objects for several different software systems. Similar to MacCormack, Rusnak and Baldwin [11], we are interested in tracking the evolution of a software system, focusing on the function call graph. In [11], their interest is understanding the impact of managerial organization on resulting software structure (primarily the modularity).

This manuscript is organized as follows. In Sec. 2 our data set is defined. Section 3 presents the top-down approach, studying evolution of global measures. Section 4 presents the bottom-up approach to understanding the relative importance of measures of structure via statistical modeling. Section 5 contains discussion and conclusions.

2 The Apache Call Graphs

We analyze the evolution of Apache for a fifty-month period using call graph snapshots taken at one month intervals from October, 2001 to November, 2005. Each monthly call-graph was created via a two step process (see [12] for more details). First, the source was checked out from the Apache CVS repository (for that month) along with matching versions of both the compiler (and associated tools) and the libraries used by Apache (*e.g.*, the Apache Portable Runtime). Then, the callgraph was extracted using CodeSurfer [13], a proprietary source code analysis tool. The resulting call graphs are directed graphs where the nodes are functions, and each edge represents an explicit call from its source node to its target node. The CodeSurfer tools extract all explicit function calls, including those to functions in libraries.

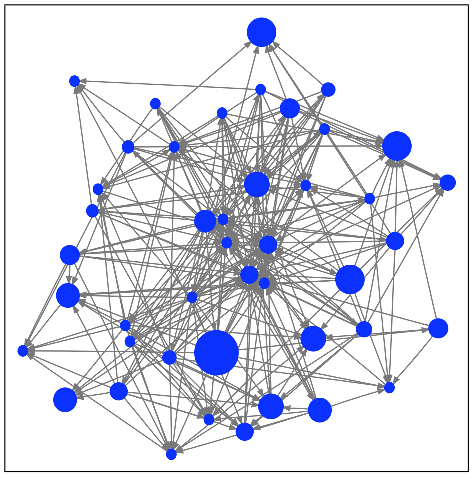


Fig. 1. The "5-core" of Apache on November 2005. Each node is a function, with size indicating its relative length in lines of code. Each directed edge is a function call.

The resulting call graphs are extremely interconnected. In November 2005, there are 2909 nodes and 8284 edges (average node degree of 5.7). The largest connected component contains all but 72 nodes, while the second largest component has only 12 nodes. Figure 1 is a subgraph showing the k-core [14, 15] at $k = 5$ for Apache functions (excluding library calls).

3 Evolution of Apache: Global measures

3.1 Nodes and edges

The most basic constituents of the Apache call graph network are the functions (*i.e.*, nodes) and function calls (*i.e.*, edges). We denote the number of functions and calls at a given time by, respectively, $N(t)$ and $E(t)$. Figure 2(a) shows their evolution over the entire 50 month period. Our first evidence for a restructuring of the code is observed during the 4th to the 5th months, when there is a dramatic decrease in N, of approximately 250 functions, accompanied by a much smaller decrease in E, of approximately 75 function calls. Thus the average degree (N/E) increases dramatically in this period.

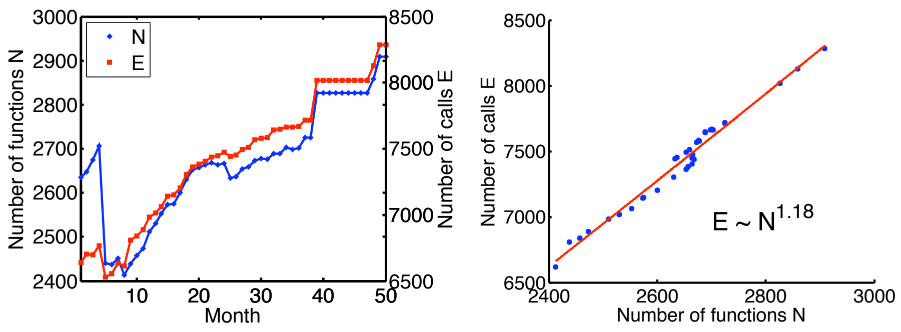


Fig. 2. (a) Evolution of the number of functions N (left-hand axis) and the number of function calls E (right-hand axis) during the 50 month period. (b) E as a function of N since the first stable release of Apache 2.0 in May 2002 through Nov 2005 (months 8-50). Dots are individual data points. The line is the best fit, $E \sim N^{1.18}$.

Investigating the Apache release history [16], we find that this period (from 2002-1-1 to 2002-2-1) marks the transition from the second to the third beta release of Apache 2.0. According to the release logs, approximately 130 changes were made to the code, with ten of these changes being addition of new features. The bulk of the remaining changes were bug fixes along with a few performance improvements. Functionality of the system was enhanced during a period where the number of functions decreased. We assume redundancy in functions was eliminated, while "functionality" (perhaps more closely related to number of edges) was preserved and enhanced.

The first stable (non-beta) releases of Apache 2.0 were issued shortly thereafter, in April and May 2002. From thereon, the relationship between E and N is extremely consistent as shown in Fig. 2(b). We find that $E \sim N^{1.18}$. Remarkably, Valverde and Solé find almost identical scaling, of $E \sim N^{1.17}$, for a collection of 80 *object oriented* systems [17], where N is the number of classes and E is the total number of edges, with each edge representing a relationship between classes. This suggests some universal trend in software systems.

3.2 Degree and degree distribution

The degree of a function conveys much information, and it is important to distinguish in-degree (being called) from out-degree (calling another function). In-degree is a measure of code reuse, and functions with high in-degree are information producers. Nodes of high out-degree are information consumers/brokers, consolidating information from many external sources. In the Apache call graphs the largest observed in-degree is approximately 200, while the largest out-degree is approximately 30. Due to these differences, we examine in- and out-degree independently.

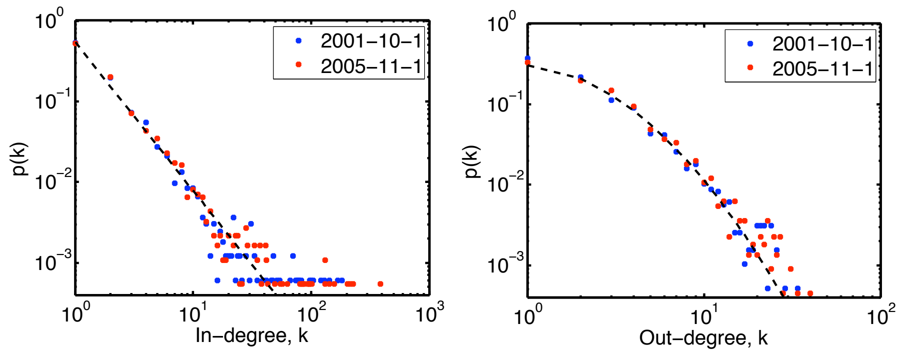


Fig. 3. (a) In-degree distribution and (b) out-degree distribution for the first month and final month. The dashed line is the best fit functional form for the final month: (a) $\mathrm{p}(k) = 0.55 \cdot k^{-1.84}$, and (b) $\mathrm{p}(k) = \left(2\pi\sigma^2 k^2\right)^{-1/2} \exp\left[\frac{-(\ln k - \mu)^2}{2\sigma^2}\right]$, with $\mu = 0.75$ and $\sigma^2 = 0.93$.

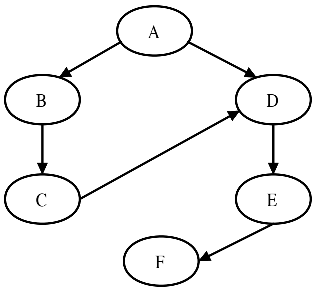
One of the most investigated aspects of "complex networks" is their degree distributions, found to exhibit extreme heterogeneity, with node degrees spanning decades of range. Here too, we find such broad-scale features. Figures 3 (a) and (b) show respectively the in- and out-degree for the first and the last of the 50 months investigated, where $\mathrm{p}(k)$ is the fraction of nodes observed with degree k.

Following [18], we asses the best fit to the data between power law, log-normal, and stretched-exponential distributions, using a weighted least squares fit. The weight given to each data point reflects inversely how much uncertainty there is in that point (more uncertainty in the tail where the values are much smaller). The quality of a fit between the set of data points $\{h_i\}$ measured a values $\{x_i\}$ and a function f is quantified as:

$$Q = \sum_{i=1}^{k} \frac{1}{h_i} \left[h_i - f(x_i)\right]^2$$

with smaller Q better. We find that, for in-degree, a power law provides the best fit for each of the fifty months with $Q \approx 0.04$. Fitting a log-normal distribution to in-degree gives $Q \approx 0.08$, and stretched-exponential gives $Q \approx 0.15$. For out-degree, log-normal provides the best fit for all fifty months with $Q \approx 0.02$. A stretched-exponential gives the next best fit with $Q \approx 0.06$, and a power law fit is the worst, with $Q \approx 0.16$.

There are small, almost indiscernible changes to the distributions over the fifty months. For in-degree we find the exponent of the best fit power law slowly decreases from $\gamma \approx 1.9$ to $\gamma \approx 1.84$, reflecting that the maximum values of in-degree slowly increase with time. For out-degree, the mean out-degree of the best fit log-normal distribution slowly increases from $\mu \approx 0.64$ to $\mu \approx 0.75$.



Dependency Matrix

	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	1	0	0
D	0	0	0	0	1	0
E	0	0	0	0	0	1
F	0	0	0	0	0	0

Fig. 4. (a) A simple call graph. (b) The equivalent dependency matrix.

However, the shapes of both the in- and out-degree distributions (power law and lognormal respectively) are global properties which are established before our data sampling begins and remain invariant throughout.

3.3 Dependencies, visibility and propagation cost

A simple call graph is shown in Fig. 4 (a). The corresponding dependency (or adjacency) matrix, Fig. 4 (b), captures the complete call graph information. Matrix element $\mathbf{M}_{ij} = 1$ if function i calls function j, and is zero otherwise. As edges are directed, $\mathbf{M}$ is not symmetric about the diagonal.

The dependency matrix captures direct dependencies. However, indirect dependencies are also important. For instance, as shown in Fig. 4 (a), a change in function C could potentially destroy or change the functionality implemented by A. Changing function F also has indirect impact on A. Yet, it is less direct, as the shortest path between A and F is of length 3, whereas the shortest path between A and C has length 2. We can quantify these indirect dependencies as a function of path length using the "reachability matrix" [19] and the related "visibility matrix" [20]. The reachability matrix at path length d, is denoted $\mathbf{R(d)}$. Matrix element $\mathbf{R(d)}_{ij} = 1$ if there is a path of exactly length d connecting function i to j. Note the convenient relationship, $\mathbf{R(d)} = \mathbf{M}^d$, where $\mathbf{M}$ is the direct dependency matrix. The visibility matrix at distance d, denoted $\mathbf{V(d)}$, is the binary sum of the reachability matrix, $\mathbf{V(d)} = \mathbf{R(1)} \vee \mathbf{R(2)} \vee \cdots \vee \mathbf{R(d)} = \mathbf{M}^1 \vee \mathbf{M}^2 \vee \cdots \vee \mathbf{M}^d$, where the operator "$\vee$" (logical or) is equivalent to binary sum. $\mathbf{V(4)}$ for our simple call-graph example is shown in Fig. 5. Matrix element $\mathbf{V(d)}_{ij} = 1$ if

Visibility Matrix with n=4

	A	B	C	D	E	F
A	1	1	1	1	1	1
B	0	1	1	1	1	1
C	0	0	1	1	1	1
D	0	0	0	1	1	1
E	0	0	0	0	1	1
F	0	0	0	0	0	1

Fig. 5. $\mathbf{V(4)}$, the visibility matrix up to path length $d = 4$ for the simple call-graph in Fig. 4 (a).

there is a path of length less than or equal to d connecting function i to j. Note we assume $\mathbf{V}(\mathbf{d})_{ii} = 1$, *i.e.*, functions are visible to themselves.

Propagation cost

Propagation cost (PC) was introduced in [11] as a scalar value to quantify the extent of indirect dependencies in a network. It is defined as the number of 1's in $\mathbf{V}(\mathbf{4})$ divided by N^2 (the total number of 1's possible). In other words, propagation cost is the number of pairs of functions connected by a path of length less than or equal to 4, divided by the number of all possible pairs. We find that changes in propagation cost (a global variable) can be useful indicators of important small-scale changes in the code base. Note we do also analyze PC for $\mathbf{V}(\mathbf{5})$, but get almost identical results.

Figure 6 shows the evolution of propagation cost, along with that of N, for the 50 months of Apache data. The baseline behavior indicates an inverse relationship (as N increases PC decreases and vice versa). There is only one region that violates this trend, encompassing months 24 to 33. Removing these months from consideration, we see an extremely consistent relation between PC and N, as shown in Fig. 6 (b), that PC $\sim N^{-0.70}$. The first anomalous event which does not confirm to this scaling relationship is month 24 (September 2003), when N decreases slightly yet PC jumps disproportionately. The second anomalous event is from months 33 to 34 (June 2004 to July 2004), when PC drops dramatically while N remains essentially constant.

No other global property discussed herein shows marked changes in this timeframe, not even during the second anomaly which is most dramatic. N and E are both essentially invariant (see Fig. 2). Degree distribution is invariant. Average clustering coefficient is invariant.

We attempt to isolate what changes in the details of Apache are responsible for these two anomalous events. Motivated by findings in [10], which

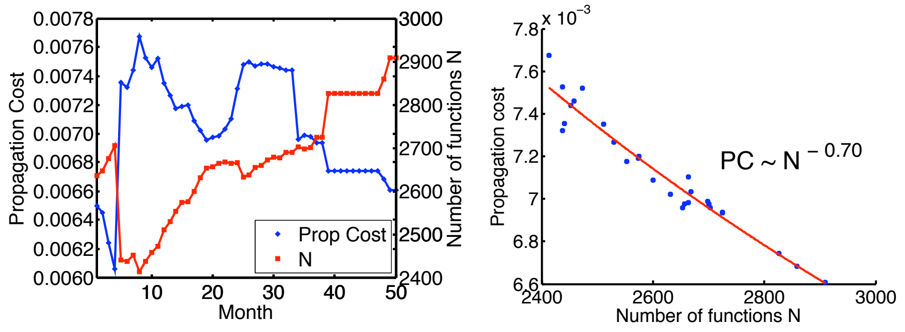


Fig. 6. (a) Propagation cost (left-hand axis) and size (right-hand axis) as functions of time. (b) PC as a function of N since the first stable release of Apache 2.0, with anomalous months (23 thru 34) removed. We find that PC $\sim N^{-0.70}$.

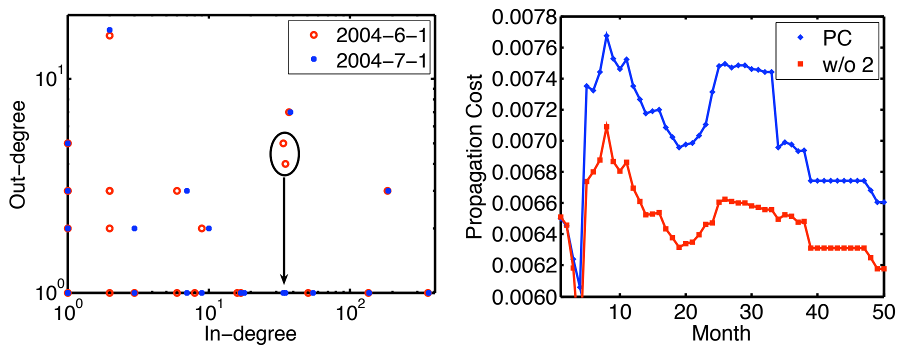


Fig. 7. (a) Scatter plot of in-degree and out-degree in log-log scale only for functions whose degree changed in this time period. (b) Propagation cost over time. Top line is for the entire system. Bottom line is resulting PC if the the two functions indicated in (a) are removed, denoted "w/o 2" in the legend.

suggest that functions with simultaneously high in- and high out-degree are particularly problematic, we isolate functions whose in- or out-degree changed during the timeframe of interest. Functions with simultaneously high in-degree and out-degree have a tremendous amount of upstream and downstream dependencies. They are simultaneously information consumers and information producers. Figure 7 (a) is a scatterplot of in-degree versus out-degree on June 2002 (circles) and July 2002 (pluses), including only functions with changes in these quantities.

Circled in Fig. 7 (a) are two suspicious functions. They have high in-degree (of 33 and 34) and reasonably high out-degree (of 5 and 4) in June 2002. They maintain the in-degree but drop, as indicated, to out-degree of one in July 2002. We remove these two functions (and their edges) from the call-graph for each of the 50 months and plot the resulting evolution of PC as shown in Fig. 7 (b). The top line is the same as Fig. 6 (a), PC for the entire system. The bottom line is the resulting PC with the two functions removed. We no longer see the anomalous behavior and recover the baseline behavior PC $\sim N^{-0.70}$ show in Fig. 6 (b).

These functions (`apr_thread_mutex_lock` and `apr_thread_mutex_unlock`) are members of the Apache Portable Runtime layer that implement functionality related to multithreading. Investigating the detailed commit logs written by developers [21], we find that on August 7, 2003 (between months 23 and 24) attempted bug fixes to these two functions were made, with accompanying comments indicating a history of problems with these two functions. On June 4, 2004 (between months 33 and 34) these two "racy/broken" functions are dropped from the code entirely and replaced with lower-level system library calls.

3.4 Path lengths, clustering coefficient and "small worlds"

A simple example call graph is given in Fig. 4 (a). There are directed paths connecting various functions. For instance, function A is connected to function F via two paths, one of length 3 and one of length 5, where length is measured by number of hops in the call graph. The path of length 3 is obviously the shortest path connecting A and F. We consider all such pairs of functions which are connected by a directed path and calculate the shortest path between them. The fraction of shortest paths of a specified length (*i.e.*, the normalized distribution) is shown in Fig. 8 (a), for the first month (October 2001) and the final month (November 2005) of our study. Similar distributions result for all 50 months, with the typical shortest path of length between 4 and 5, and the largest shortest path (*i.e.*, the graph diameter) of length 14.

We compare this distribution of shortest paths to those resulting from two different random graph growth processes. First we consider an ensemble of 20 realizations of Erdős-Rényi random graphs [22, 23] with $N = 2909$ nodes and $E = 4142$ undirected edges (equivalent to the $N = 2909$ nodes and $E = 8284$ directed edges in the November 2005 Apache call graph). Here we find the typical shortest path is of length 7 or 8, much larger than for the Apache call graphs. However, the diameter is comparable, ranging from length 14 to 16.

The degree distributions of the Apache call graphs (see Fig. 3) are much broader and more heterogeneous than the Poisson distribution which characterizes Erdős-Rényi random graphs [22, 23]. Thus we next compare the Apache graphs to random graphs constructed to match exactly the Apache degree distribution by extending the ideas in [24, 25] to directed graphs. We begin with $N = 2909$ nodes and map each one to a distinct node in Apache. We assign to each of these new nodes the in- and out-degree of their corresponding Apache node. We do not yet specify the connectivity, only the final degree. In other

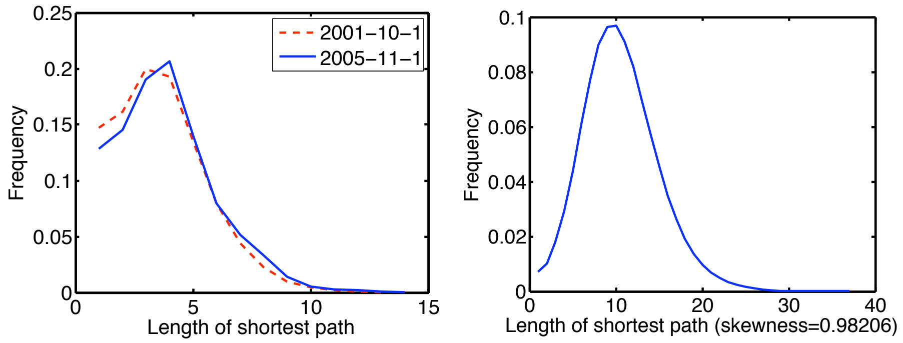


Fig. 8. (a) Normalized shortest paths in Apache, first month and last month. (b) Normalized shortest paths averaged over 20 realizations of random networks with the exact in- and -out degree distributions of Apache on November 2005. The vertical axis, "frequency", means the fraction of shortest paths having that length.

words, we assign unconnected half-edges. We next perform a random matching and pair up each in-degree half-edge with a different out-degree half-edge chosen at random. We construct an ensemble of 20 such random graphs. The resulting normalized shortest path distribution, averaged over the full ensemble, is shown in Fig. 8 (b). Note, the typical path length is much larger than for Apache, peaking at length 10, and the maximum shortest path is around 30. Matching degree distribution alone is not enough to reproduce the shortest path lengths observed for Apache.

"Small world" networks are characterized by small diameters and large clustering. We have established the small diameter above. Throughout the 50 month period the average clustering coefficient, C, fluctuates in the range $0.09 < C < 0.099$. Calculating C over an ensemble of corresponding Erdős-Rényi random graphs yields $C = 0.0018$ and for the ensemble of random graphs with the Apache degree distribution $C = 0.023$. The Apache call graphs thus have the "small-world" characteristics of short average path length and relatively large clustering coefficient when compared to a comparable random graph. Note, to measure C we temporarily assume the edges are undirected. A more thorough treatment is presented in the next section, where "transitive" triads are distinguished from "cyclic" triads. (Cyclic triads are rarely seen in software, though transitive ones occur frequently.)

4 Evolution of Apache: Models of Network Structure

Above, we made a number of empirical observations about the Apache call graph using complex network measures, effectively obtaining a multifaceted characterization of the graph. One can ask, how do these, and possibly other, measures combine together to tell the story of the whole Apache call graph? And in general, to what extent is its structure determined by any given observations?

To answer these questions, here we present the statistical modeling approach of *Exponential Random Graph Models (ERGMs)*, developed in recent social network theory [26, 27] for understanding the relationships between a large class of *local* network observations and the full network structure. This bottom-up approach models the extent to which a set of specific observations (*e.g.* counts of transitive triads) explain the global structure of a network (*e.g.* the Apache call graph), and, in the process, determines which of the observations best explain its structure. More specifically, given a set of observations, or explanatory variables, an ERGM models networks as random samples from an exponential probabilistic space given by linear combinations of those explanatory variables. Thus, given a network and fitted ERGM, one can calculate the probability that the network is determined by those variables, via direct calculations. In practice, these models are very appealing as there exist methods for both model fitting (observation available) and simulations (observations unavailable).

The advantage of the ERGM approach is that it is very general and scalable; the architecture of the graph is represented by the chosen set of explanatory variables which can describe either local or global features of the network, and the values of the model parameters can be quite instructive, indicating the relative importance of the explanatory variables to the maximum likelihood pdf. In addition, ERGMs have been well studied and theoretical results exist which can offer some understanding of the model's behavior in practice [27].

4.1 ERGM Theory

Here, we describe formally the ERGM statistical framework for modeling networks, in particular as it pertains to modeling software call graphs. Let $\boldsymbol{X}$ be a random variable representing the adjacency matrix of a software network. The probability distribution function (pdf) for this random variable, $P(\boldsymbol{X} = \boldsymbol{x})$, tells us the probability that an observed graph, $\boldsymbol{x}$, was drawn from $\boldsymbol{X}$. Unfortunately, the probability distribution function of $\boldsymbol{X}$ is unknown and cannot be directly calculated. To estimate this pdf, let $\boldsymbol{z}(\boldsymbol{x}) = (z_1(\boldsymbol{x}), z_2(\boldsymbol{x}), \ldots, z_r(\boldsymbol{x}))$ be a vector of explanatory variables, where each explanatory variable can be any function of the observed data. We postulate that there exists $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_r)$ such that:

$$log(P(\boldsymbol{X} = x)) \propto \theta_1 z_1(\boldsymbol{x}) + \theta_2 z_2(\boldsymbol{x}) + \ldots + \theta_r z_r(\boldsymbol{x}) \propto \boldsymbol{\theta}^T \boldsymbol{z}(\boldsymbol{x}) \tag{1}$$

If we exponentiate both sides and divide by a normalizing constant, $\kappa(\boldsymbol{\theta})$, assuring that the probabilities will sum to one, we get the following model:

$$P(\boldsymbol{X} = \boldsymbol{x}) = e^{\boldsymbol{\theta}^T \boldsymbol{z}(x)} / \kappa(\boldsymbol{\theta}) \tag{2}$$

This is the standard log linear probability model that is used in a wide range of fields from the social sciences to biology [28, 29].

To create an ERGM, a set of explanatory variables (virtually any function from the observed graph to the real numbers) is chosen by the modeler. The choice of variables is based on the pertinent features of the graph under study, or on a set of desired features, if the graphs are being simulated. An example, non-exhaustive, set of explanatory variables is given in Table 1, most of which are important for modeling the Apache call graph. The coefficients, $\boldsymbol{\theta}$, can be interpreted as a preference of the observed network for a given explanatory variable, if its coefficient is positive, and a preference against a variable, if it is negative.

Estimating $\boldsymbol{\theta}$ based on an observed network is referred to as fitting the model, while using a predetermined $\boldsymbol{\theta}$ to generate networks is referred to as simulating with the model. Given a set of explanatory variables, the best fit to the observed network is given by the parameter vector $\boldsymbol{\theta}$ which maximizes the likelihood that the observation is drawn from the probability distribution given in Eq. 2. In this case, though, the standard maximum likelihood

Variable	**Description**
`istar(`k`)`	The number of k-tuples of edges that point to the same node in the network.
`ctriad`	The number of 3-cycles in the network.
`ttriad`	The number of two edge paths for which there is a one edge shortcut in the network.
`triangle`	The sum of `ctriad` and `ttriad` for the network.
`idegree(`k`)`	The number of nodes with exactly k incoming edges in the network.
`odegree(`k`)`	The number of nodes with exactly k outgoing edges in the network.
`gwidegree`	The sum of the counts of each in-degree, weighted by the geometric sequence, $(1-e^{-\theta_k})^i$ where θ_k is a decay parameter.
`edges`	The number of edges in the graph.

Table 1. Exponential random graph models are extremely flexible. This table shows several example explanatory variables, identifying the variables by their names in the `statnet` package for R [30].

method to estimate the parameters is difficult because the function for the normalizing constant $\kappa(\boldsymbol{\theta})$ is not known a priori. Instead, one typically uses Markov chain Monte Carlo maximum likelihood estimation (MCMC MLE), a family of methods based on the Newton-Raphson MLE algorithm [26]. The maximum likelihood formula for the pdf obtained via fitting can be used with Markov chain Monte Carlo (MCMC) sampling methods to simulate networks. There are a number of software packages available for MCMC MLE fitting. These include the "statnet" package [31] for R and the stand-alone SIENA software [32].

In practice, one rarely knows which explanatory variables to chose to fully describe a network using ERGMs. To compare if a particular set of explanatory variables models an observed network better than another, one can use several different approaches. For example, the modeler can use the fitted model to simulate a suite of networks and check how well the simulated networks match the observed network on any measure of interest (*e.g.* the degree distribution). Along this line, the "statnet" statistical package has a built-in goodness-of-fit function which compares simulated networks to the observed network on a set of such measures. Another approach for comparing sets of explanatory variable is to use information-theoretic measures, like the Akaike Information Criterion (AIC), to assess how well a model fits the observed data. In addition to providing information on the goodness of fit, the AIC (which penalizes more complicated models to protect against overfitting [33]) can also be used to guide the search through the space of possible models, helping to identify the best variables to include in the model as follows: If the modeler suspects that a particular variable might be useful in modeling an observed network, the AIC can be used to test this hypothesis by toggling the variable in and out of the model, accepting the hypothesis if significant improvement in the AIC is observed.

4.2 Modeling Process and Results

As an exploratory first step to our modeling process, we fit models made from many of the possible combinations of a diverse set of explanatory variables that we expect to be important in explaining the Apache callgraph. We include the counts of connected triads (`ctriad` and `ttriad`, *cf.* Table 1) in many of our exploratory models because these small connected graphs (*graphlets*) may be important architecturally in many types of larger networks [34, 35]. However, we do not expect the `ctriad` graphlet to be helpful in modeling software because it implies indirect recursion, an uncommon and difficult programming technique, but we include it in our modeling process as a sanity check. We also investigate various in- and out-degree counts because these counts provide a local measure of the network's topology. Further, the success of in-degree count as an explanatory variable leads us to investigate the related in-star variables. In previous modeling efforts [36], degeneracy in the fitting algorithms was often observed for models using the variables above. To circumvent such degeneracies it has become standard ERGM practice to include the geometrically-weighted in-degree distribution and the simple edge count, as variables in every model, and we do so here too.

Model	**AIC**
`edges+gwidegree`	104090
`edges+gwidegree+ctriad`	104088
`edges+gwidegree+ttriad`	101473
`edges+gwidegree+ttriad+odegree(2)`	100065
`edges+gwidegree+ttriad+istar(3)`	97723
`edges+gwidegree+ttriad+idegree(2)`	97589
`edges+gwidegree+ttriad+istar(2)`	94383
`edges+gwidegree+ttriad+idegree(2)+idegree(3)+istar(2)`	91017
`edges+gwidegree+ttriad+idegree(2)+idegree(3)+istar(2)+istar(3)`	89491

Table 2. The Akaike Information Criterion for a sample of fitted models. Note: For space and readability, the notation we use here to describe the models omits the θ_i parameter coefficient from Eq. (1). Each term (seperated by '+') is a seperate model predictor variable with its own coefficient.

We toggle the variables described above in and out of several models, identifying variables that are important in fitting the Apache call graph to a single representative month (June, 2003). The results for several representative models are given in Table 2, together with the AIC for the model. As expected, the AIC changes very little when `ctriad` is added to the basic `edges+gwidegree` model, indicating the lack of importance of `ctriad`, but the AIC improves significantly when the `ttriad` variable is added, showing us that the tendancy of Apache programmers to include layer-crossing function calls is important in determining the global nature of the graph. Given these results, we further refine our search, looking at many more models that include the `ttriad` variable, and we find that the outdegree and the higher indegree terms are less important than others we consider.

Table 2 allows us to see the variables that are important to the AIC and, hence, are better at predicting the topology of the apache callgraph. For example, it is interesting that the out-degree of a function is less important to the global topology than the in-degree, indicating that the emergent structure of the callgraph is more dependent on how many times each function is called rather than how many dependencies they have, which is in line with the findings in Sec. 3.

Next, we perform a longitudinal, 50-month study of the Apache callgraph using a few of the best fitting models from the one-month study. This experiment lets us see if the relative importance of explanatory variables changed throughout the apache development process. The ranking by AIC of the models we fit remains constant across all 50 months, but the values of the parameters do not. Fig. 9 shows a plot of the coefficient values over time for `ttriad`, `idegree(2,3)` and `istar(2,3)`. These variables were chosen because they were contained in our best fitting model (as determined by AIC) from Table 2, and we chose not to study any variables (such as `odegree`) from other, less well-fitting models. Our exploratory procedure eliminated the other variables that we considered because they did not contribute as large an improvement to the AIC as the variables from the final model.

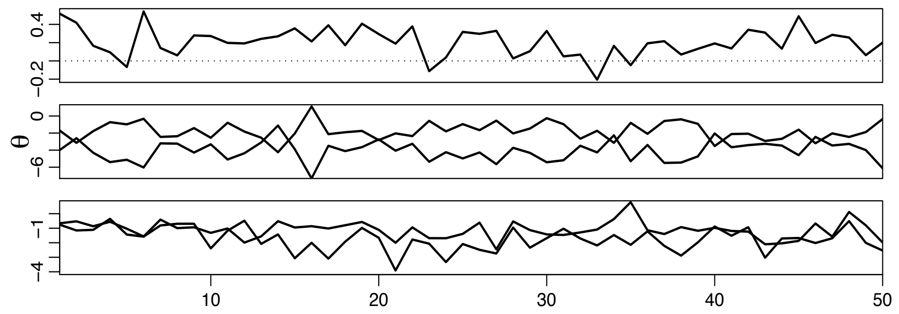


Fig. 9. Plots of several interesting coefficients across all fifty months. Top: `ttriad`. Middle: `idegree(2,3)`. Bottom: `istar(2,3)`.

All of the variables that we've measured relating to in-degree (`idegree(2,3)`, `istar(2,3) and gwidegree`) are generally negative in this model. On the other hand, the transitivity variable `ttriad` is consistently positive throughout the development cycle. This indicates that there are functions in Apache that call their callee's callees (perhaps due to the standard library functions being included in the apache callgraph).

Interestingly, over the 50 month period, `indegree(2)` is almost perfectly anti-correlated with `indegree(3)` (as seen in Fig. 9). One explanation is that these two variables are measuring two aspects of the same phenomenon (how many functions are called approximately twice), and, hence, the importance of the two variables to the model is correlated. Similarly, `edges` and `gwidegree` (not shown) are strongly anti-correlated, perhaps, because they both measure aspects of network density.

5 Discussion and Conclusions

We study the evolution of the function call graph for the Apache 2.0 HTTP server over a fifty moth period. Apache is a mature, open source software project, written in a procedural programming language. We characterize Apache first with several global measures, 1) nodes and edges, 2) degree distribution, 3) dependency matrices and propagation cost, 4) path length and clustering. We find these measure have certain baseline behaviors and that deviations can indicate important structural changes in the code base. In particular, we find that propagation cost (introduced in [11]) is a sensitive measure that can signal when a detailed, fine-grained examination of the code base may be required. Using ideas proposed in [10] (that functions with simultaneously high in- and out-degrees are problematic), we are able to isolate that the large changes observed in propagation cost are attributable to just two individual functions (out of approximately 2900 total functions). By examining the detailed development logs we corroborate that indeed these two functions have repeatedly troubled developers. The techniques presented herein may be useful in general for code written in procedural programming languages as they may allow developers to identify particular functions which, when restructured, can reduce overall system dependencies.

Using exponential random graph modeling, we investigate the relationships between the attributes that we empirically observe, and find that the most important attribute for predicting the global structure of the Apache callgraph is `ttriad`, the number of transitive triads in the graph. In future work we intend to explore how the appearance of unexpected features might help to identify bugs.

6 Acknowledgments

We are indebted to Christian Bird for supplying the call graph data which is central to our analysis and to Premkumar Devanbu for many useful discussions. This work was funded in part by the National Science Foundation under Grant No. IIS-0613949.

References

1. E. S. Raymond. *The Cathedral & the Bazaar.* O'Reilly and Associates, Sebastopol, CA, 1999.
2. T. O'Reilly. Lessons from open source software development. *Communications of the ACM*, 42(4), 1999.
3. P. Ball. Openness makes software better sooner. *Nature*, June 25, 2003.
4. D. Challet and Y. Le Du. Microscopic model of software bug dynamics: Closed source versus open source. *International Journal of Reliability, Quality and Safety Engineering*, 12(6), 2005.
5. M. Fowler. *Refactoring: Improving the Design of Existing Programs.* Addison-Wesley, 1999.
6. A. A. Gorshenev and Yu. M. Pis'mak. Punctuated equilibrium in software evolution. *Phys. Rev. E*, 70(6):067103, 2004.
7. http://httpd.apache.org.
8. See for instance, Software Maintenance Costs and references therein, http://www.cs.jyu.fi/∼koskinen/smcosts.htm.
9. S. Valverde, R. Ferrer Cancho, and R. V. Solé. Scale-free networks from optimal design. *Europhys. Lett.*, 60(4):512–517, 2002.
10. C. R. Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Phys. Rev. E*, 68:046116, 2003.
11. A. MacCormack, J. Rusnak, and C. Y. Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(7), 2006.
12. Zachary M. Saul, Vladimir Filkov, Premkumar T. Devanbu, and Christian Bird. Recommending random walks. In *ESEC/SIGSOFT FSE*, pages 15–24, 2007.
13. http://www.grammatech.com/products/codesurfer/overview.html.
14. S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5:269–287, 1983.
15. B. Bollobas. The evolution of sparse graphs. In *Graph Theory and Combinatorics*, pages 35–57. Academic Press, 1984.
16. http://www.apacheweek.com/features/ap2#rh.
17. S. Valverde and R. V. Solé. Hierarchical small worlds in software architecture. In *Dynamics of Continuous Discrete and Impulsive Systems: Series B; Applications and Algorithms*, volume 14, pages 1–11, 2007.
18. G. Baxter M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero. Understanding the shape of java software. In *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 397–412, New York, NY, USA, 2006. ACM.
19. J. N. Warfield. Binary matrices in system modeling. *IEEE Transactions on Systems, Man, and Cybernetics*, 3:441–449, 1973.
20. D. Sharman and A. Yassine. Characterizing complex product architectures. *Systems Engineering Journal*, 7(1), 2004.
21. http://svn.apache.org/viewvc/.
22. P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.
23. P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci.*, 5(17), 1960.

24. M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Alg.*, 6:161–179, 1995.
25. M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64:026118, 2001.
26. Tom A. B. Snijders. Markov chain monte carlo estimation of exponential random graph models. *Journal of Social Structure*, 3(2), 2002.
27. Caroly J. Anderson, Stanley Wasserman, and Bradley Crouch. A p* primer: logit models for social networks. *Social Networks*, 21:37–66, 1999.
28. David Kaplan. *The Sage Handbook of Quantitative Methodology for the Social Sciences*. Sage Publications Inc., 2004.
29. Claire Infante-Rivard, Clarice R. Weinberg, and Marguerite Guiguet. Xenobiotic-metabolizing genes and small-for-gestational-age births: Interaction with maternal smoking. *Epidemiology*, 17(1):38–46, 2006.
30. The R Project for Statistical Computing, http://www.r-project.org.
31. Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. statnet: An r package for the statistical modeling of social networks, 2003. `http://www.csde.washington.edu/statnet`.
32. Tom A. B. Snijders, Philippa E. Pattison, Garry L. Robins, and Mark S. Handcock. New specifications for exponential random graph models. *Sociological Methodology*, 2006.
33. Konishi, Sadanori, Kitagawa, and Genshiro. *Information Criteria and Statistical Modeling*. Springer, 2008.
34. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298:824–827, 2002.
35. S. Valverde and R. V. Solé. Network motifs in computational graphs: A case study in software architecture. *Phys. Rev. E*, 72:026107, 2005.
36. D. R. Hunter and M. S. Handcock. Inference in curved exponential family models for networks. Technical report, Penn State Department of Statistics, 2004. Available from `http://www.stat.psu.edu/reports/2004/`.