Classifying Network Complexity

Michael H. Behringer
Cisco Systems
mbehring@cisco.com

ABSTRACT

Growing complexity in the Internet is the subject of many debates. While there is extensive research on complexity in some specific areas such as graph theory or software design, the complexity of a real life network is not very well defined or understood. This position paper provides a proposed classification of network complexity, some observations, and an overview of approaches to control network complexity. The goal is to stimulate and aid future discussions on the subject.

Categories and Subject Descriptors: C.2.1 [Computer Systems Organization]: Computer-Communications Networks, *Network Architecture and Design*

General Terms: Design, Reliability, Security

Keywords: Internet complexity; network complexity; simplicity; robustness.

1. PROBLEM STATEMENT

We observe an increase in a number of parameters on the Internet. Examples are the number of prefixes and autonomous systems in the global routing table [CIDR-Report]. Similar trends can be observed in many areas. For example, the size of length of router configurations (figure 1) displays a linear growth curve. As intuitively expected, edge routers grow faster than core routers. The likely reason is that customer specific services are configured at the edge, making edge configurations longer.

Also the size of operating systems is growing over time. Figure 2 illustrates the growth of the binary executable for one router's operating system. Similar trends can be observed on other operating systems.

ReArch'09, December 1, 2009, Rome, Italy.

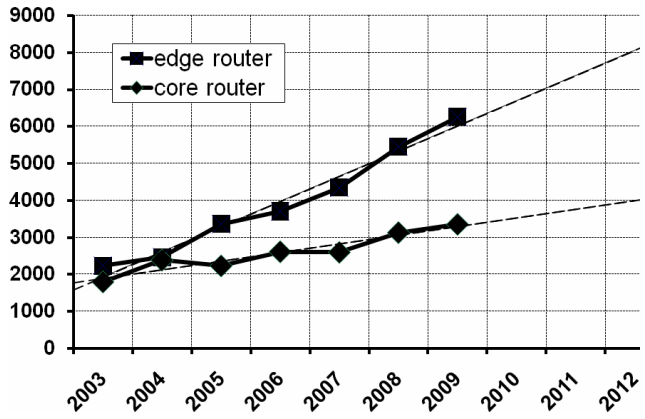

Figure 1: Configuration size (lines) of two typical routers at a tier-2 service provider.

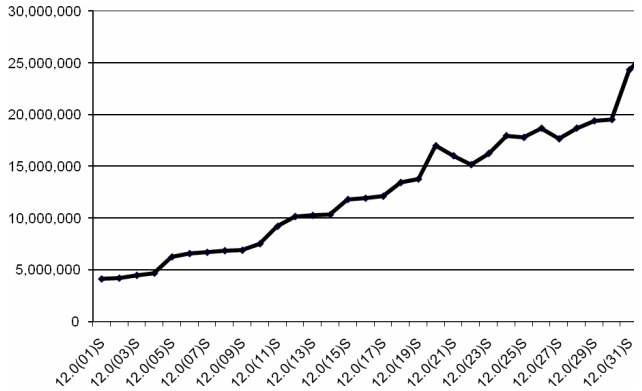

Figure 2: Increase of code size of Internet routers. (Example: Size of binary (bytes), Cisco 12.0S train)

Even without a precise definition of "complexity", it seems intuitive that the Internet is becoming more complex. It is however less clear which impact the various part of this growing complexity will have in the future. For example, the increasing size of router configurations may reflect a complexity increase if it is due to added features and functionalities; if it is due to iterative replication of interface configurations it may however not raise complexity significantly.

The growth in network operating systems size is an example of a trend which can be observed in many areas: There is a general tendency to add new features and functionalities; however, rarely is existing functionality removed. This is mainly due to the uncertainty whether such features are still

in use or not, and the related cost of removal. This applies to many parts of the network, for example the operating systems, the number of protocols used on the network, or packet filters and firewalls.

Observation #1: In the absence of radical re-organization, most systems in the Internet tend to grow, never shrink.

We believe that with the number of features increasing with every release the complexity of routers, firewalls, and other network systems increases as well, both on the operating system (see figure 2) , as well as the configuration (figure 1). While in the early days of networking all operators in the NOC understood the entire configuration of a router, today, with increasing specialization, many operators only understand completely the parts they are specialized in. As a corollary fewer operators understand the entire feature set and configuration of a network device. We therefore note:

Observation #2: The complexity of operating a system is inversely proportional to the number of people who understand the system in detail.

Complexity inversely affects predictability of systems. For example the configuration of filter lists is complex in the sense that in reality few people understand the reasoning behind all the rules. Therefore, it is often not entirely predictable what the outcome of a rule change will be. The same is true for many configuration changes in networks: It is often hard to predict the precise effect of a change. Simulation tools are often used to model this uncertainty.

Since predictable network behavior is desirable, methods are required to determine the level of complexity, and to control complexity. This paper suggests a way of breaking down and classifying network complexity.

2. PREVIOUS WORK

A network can be depicted a graph, and routing protocols fundamentally treat the network as such. There is ample research on graph complexity, which has been defined in many different forms, for example by the number of its spanning trees [Biggs; Constantine; Grone], or the linear complexity of any one of the graph's adjacency matrices [Neel].

Work on software complexity applies directly to the network operating system. It is measured in the number of functions or modules, their dependencies, and the number of execution paths. There is a wide range of research available in this field, dating back to the 1970s [McCabe].

However, intuitively the complexity of a real life network does not only depend on vertices (nodes) and edges (links) of the network, nor on the complexity of the operating systems. The complexity of a network also depends on the configuration, the rate of change, and other dynamic properties. David Meyer has spent considerable effort in compiling information and researching on network complexity [Meyer], however without defining in detail how to measure complexity.

John Doyle´s research on complexity [Doyle] is also directly applicable to networks. He coined the "robust yet fragile" paradigm, and correlates robustness and complexity, depicted in figure 3.

It is desirable for a network to be robust, to be available when needed. However, unpredictable circumstances, such as a fiber cut, can arise. Therefore, the network must have a way to re-route traffic to a working path. Re-routing however requires a certain amount of complexity, which must be provided in the overall network.

Observation #3: Robustness requires a certain amount of complexity.

Bush and Meyer describe a number of network related complexity guidelines [RFC3439], including the principles of amplification and coupling. They deduce some architectural guidelines on how networks should be engineered, and also refer to the "simplicity principle", first coined by Mike O'Dell: "Complexity is the primary mechanism which impedes efficient scaling, and as a result is the primary driver of increases in both CAPEX and OPEX."

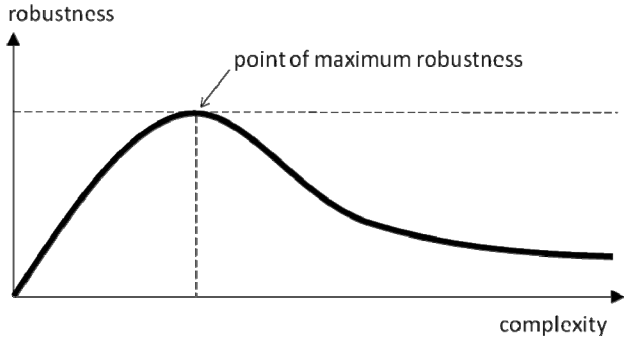

Figure 3: Robustness / complexity distribution [Doyle]

Given observation #3, we note that a minimum level of complexity is required for a robust network. We therefore refine the simplicity principle by replacing "complexity" with "unnecessary complexity". It is however beyond the scope of this paper to define what is necessary.

3. A CLASSIFICATION OF NETWORK COMPLEXITY

3.1 A High-Level Model

Starting from a highly abstract view, we define complexity in networks as a function of state and rate of change over the components of a network. We believe that dependencies, number of execution paths and other common complexity metrics can ultimately be represented by state. For example, routers have dependencies on their neighbors, but this is expressed in the configuration of routing protocols, and

routing tables. Different elements of state may not be directly comparable, and may require some "complexity weight" to express their relative importance to complexity. For example, an external BGP peering may carry a different weight with respect to complexity than an internal one.

Since human error is a major factor in successfully running and maintaining networks, we also include the network operator as a critical part in this observation. Network management systems also play a role in modern networks. Since they can abstract complexity from the operator we also include them in this model. Figure 4 depicts the elements of a network considered in this paper.

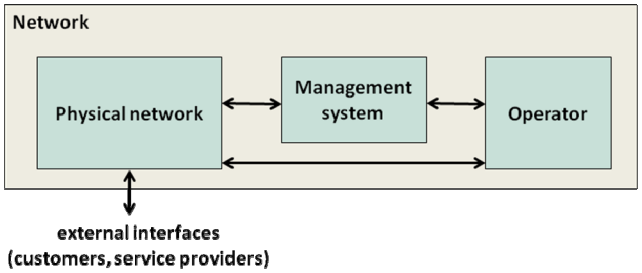

Figure 4: Elements of a network

High-level dependencies are represented here as arrows. Depending on the type of network, each component can be more or less complex. This depends largely to the amount of state of each element. As explained above, state also represents internal dependencies within each element and with other elements.

The physical network contains:

- Network devices, such as routers, switches, optical equipment, etc. This includes components in those devices, such as CPUs, memory, ASICs, etc.
- Links between devices.
- External links, to customers and other service providers.
- Support hardware, such as power supplies, heating, cooling, etc.
- Operating systems.
- Device configurations.
- Network state tables, such as routing tables, ARP tables, etc.

The management system consists of:

- Hardware used for the management systems, and the network connecting them.
- Operating systems.
- Software for management, provisioning, etc.
- Operational procedures.

The operator is an abstract notion for the combined knowledge required to operate the network. We note:

Observation #4: The people operating a network, including their collective knowledge required to do so, represent state which needs to be accounted for in complexity metrics.

Strictly speaking, network management systems are not required, and operators could directly configure the network elements. The addition of the network management system apparently increases the complexity of the overall system. However, in a well designed system, the operator now has less direct interaction with the network, which reduces the likelihood of human error, and thus increases the predictability of the system. Following the above observation, the overall state of the "operator" can be decreased, either by using less skilled, or fewer, operators. Therefore, the complexity of the network is not necessarily going up with the introduction of a network management, instead it shifts complexity away from the operator to the network management system.

Observation #5: Growth in complexity on the network management side generally reduces operator complexity, and potentially the complexity of the overall system.

Human intervention has its own set of problems; the most relevant relating to complexity is their unpredictability. While a well-designed algorithm and network management has a fairly predictable behavior, the behavior of a human is never entirely predictable. This introduces a level of uncertainty into the network, which cannot be reliably modeled, directly increasing the complexity of the overall network.

Observation #6: It is generally preferable to move complexity to the component that needs less human interaction.

Operational procedures are a practical way to make human behavior more predictable, by defining clear guidelines on how to proceed in certain situations. The procedures themselves however are part of the overall system and thus increase its complexity.

All of the above components are subject to change, and the rate of change is another parameter that influences the complexity. Hardware and operating system changes are relatively infrequent, configuration changes more common, and state changes in the network extremely frequent. A high number of changes leads to coupling and amplification effects, as explained in RFC 3439 [RFC3439].

3.2 Visualizing Complexity as a Cube

Resuming the previous example of re-routing, there are various ways of implementing it: The current approach in the Internet is to let protocols deal with this task. This implies a certain protocol complexity, which is part of the operating system of the routers, and contributes to the complexity of the physical network. Another way to provide re-routing, used by traditional telephony companies, is to keep the network elements simple, and let the management system

take care of re-routing. In this case, the complexity of this task is with the network management, and the physical network is less complex than in the previous case. It is also possible to let the operator take re-routing decisions, in which case the complexity moves to this component. Figure 5 illustrates this principle.

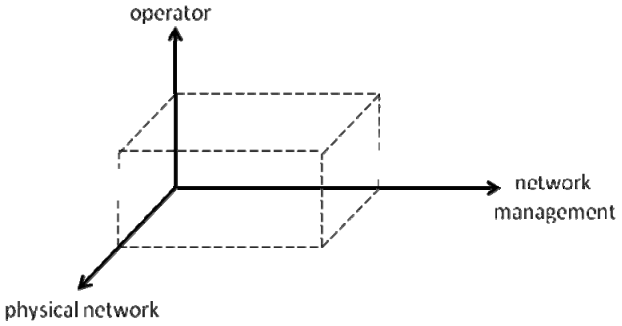

Figure 5: The complexity cube displays how single components influence overall network complexity

The overall complexity of a network is composed of three vectors: the complexity of the physical network, of the network management, and of the human operator. The volume of the cube represents the complexity of the overall network. External links, shown in figure 4, can be counted as part of the physical network.

Using this model, an Internet provider network in the beginnings of the Internet would have a complexity cube similar to the one depicted in figure 6. In the early days of the Internet configurations were mainly carried out directly by the operators, with little use of network management systems. The complexity of the physical network was relatively smaller compared to today, due to less features.

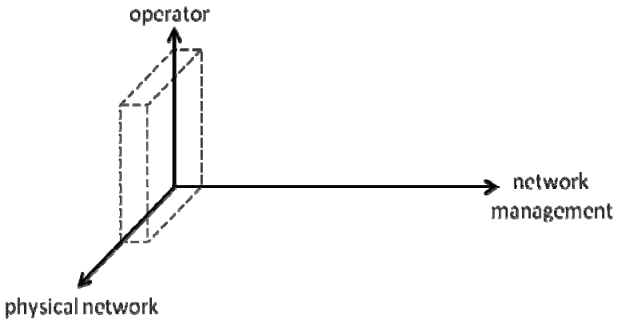

Figure 6: Complexity cube for an early Internet provider

Large service providers today attempt to lower the dependencies of human operators, and instead use sophisticated management systems. An example complexity cube could look like illustrated in figure 7. Overall complexity of today's networks, illustrated by the volume of the cube, has increased over the years. This leads to the following observation:

Observation #7: Complexity can be shifted between the components of a network.

Note that the axis of the above graph may not share the same scale; therefore moving a problem from one component to another is likely to change the overall complexity, in either direction.

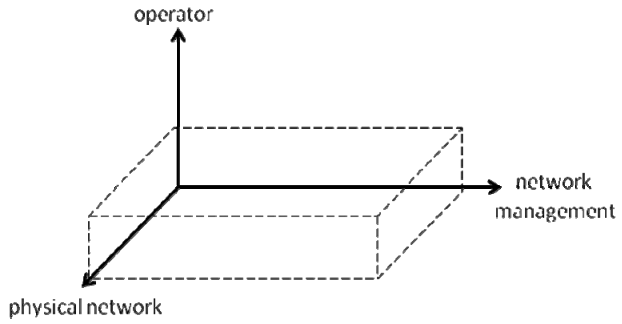

Figure 7: Complexity cube for a large Internet provider

3.3 Dependencies in a Network

While the previously defined high-level view may help to illustrate certain points, it is apparent that complexity needs to be defined at a much finer scale, especially if the goal is to eventually provide quantitative metrics for network complexity. We therefore need to break up each of the high level components into smaller more manageable pieces. As an example, we look at the complexity of the physical network, some of the components of which are listed in the previous section.

Again, within the physical network we observe the same dynamics of complexity: For example, a route can be derived from a configuration (static route), in which case the complexity is in the configuration. This also implies some complexity on the operator side, who needs to know the desired routing. It could also be derived from a protocol, in which case the complexity is in the routing protocol, and indirectly in the operating system implementing it. Figure 8 depicts a simplified containment graph of a network.

The components of this graph are highly dependent, and not only within a sub-tree, but across the entire graph. For example, adding a link between two routers affects many components (depicted in dashed boxes), such as the configuration, the hardware of the connecting routers, protocol parameters, and potentially even elements like access control, in the case where security configurations prevent access from certain interfaces. The network management must model the network and thus is also likely to reflect the change.

There is ample applied and fundamental research on the subject of network modeling, for example [Kelly; Strassner; Quoitin].

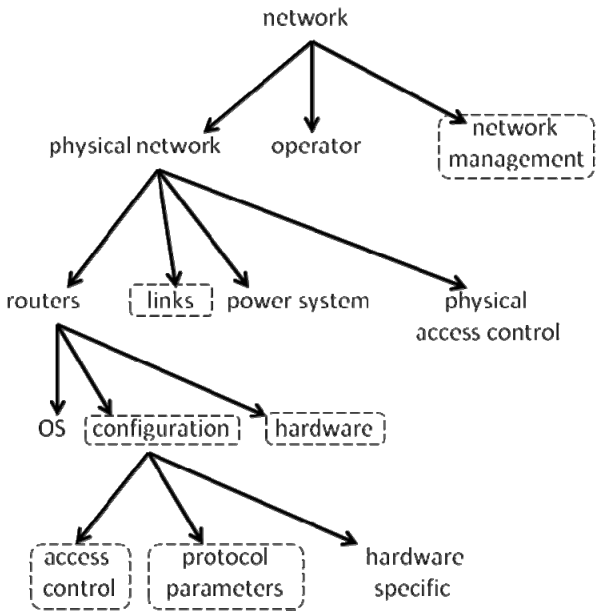

Figure 8: Containment graph of a network

There are also commercial and free tools available for this purpose. Most however focus on specific parts of the network, for example the routing design. More work is required to model and understand dependencies in real-life networks, especially including the human element.

4. APPROACHES TO DEALING WITH COMPLEXITY

4.1 Divide and Conquer

The most common way to deal with a complex issue is to break it up into smaller, more manageable and less complex pieces. There are various ways of achieving this:

4.1.1 Layering

Layers hide complexity, by offering a simplified interface to the layer above. Thus a layer does not need to understand all details of the other layers. However, often this approach has a performance penalty; also functionality abstracted away in a lower layer may sometimes be advantageous to the upper layers. For example, the IP layer doesn't know about paths on the physical layer. Human intelligence is therefore required to construct truly path redundant networks. Another issue with layering in networks is that similar decisions may be made at various levels of the stack, asynchronously. A link outage for example will be detected by several layers, asynchronously. This can lead to unexpected race conditions.

4.1.2 Object-Oriented Approaches

Defining classes and instantiating objects from those classes also provides a simplification. Also here complexity is hidden in the class, and the user of an object has a unified interface to it. Note that the class concept can be applied on many different levels of a network. Examples are the configuration (for example, defining a class for a certain interface type), router (for example, defining hardware and software for a type of customer premises equipment), or even a point of presence in a service provider network (defining type of routers, cabling, interface numbering schemes, etc).

A service provider, who keeps for example all customer facing interfaces in a small set of classes, defining for instance hardware type, configuration, security settings, and operational procedures, can apply the same rules to all interfaces in the same class, greatly simplifying operations. This does however make the service more rigid, since exceptions should be absolutely minimized, and themselves ideally handled in exception classes.

Observation #8: The complexity of a system depends to a large extend on the number of classes defined, not of their instantiations.

4.2 Shifting Complexity

In section 3.1 we introduced some examples for shifting complexity between components of the network, which is another way to control complexity. Since human error is the most important reason for system failure, hiding complexity from the users and operators is an important approach to manage complexity. User-friendly operational interfaces do not actually reduce the complexity of the system, but they constrain user input, thereby increasing predictability, while reducing the chance of errors. The overall experience is better than with a poor user interface, which allows for more errors. Effectively a simple user interface shifts complexity from the operator, which generally reduces overall complexity (see observation #6).

4.3 Meta Languages

Another way to reduce human error is to provide simplified meta languages, hiding internal complexity in simplified commands. Since such languages could also be standardized between different vendors, management of multi-vendor networks would be greatly simplified. Examples of such configuration and policy meta languages are the Common Information Model [CIM], NetConf [RFC4741], and the Routing Policy Specification Language [RFC2622].

4.4 Structural Approaches

Complexity is related to system dependencies. Structural approaches search analytically for dependencies, and try to find ways to reduce them. This has been researched primarily for software (e.g., [Sangal]), but the concept is equally applicable to networks. Such approaches truly reduce the complexity of a solution; however, as explained in section 3.3, real life networks are difficult to model for dependencies.

5. SECURITY CONSIDERATIONS

Specifically when discussing the security of systems, the need for predictability becomes apparent: In a network we expect a deterministic decision on whether a packet or flow is forwarded or dropped. With increasing complexity this decision cannot be made without incurring false positive and false negative decisions. And while on a packet level a decision may be deterministic, this is generally not the case on the application level. For example, packets from a client to a server can be blocked; however the user may bypass this rule, for example by connecting from a different host.

Observation #9: The more complex the network, the more likely a security rule can be bypassed.

It is commonly stated that "complexity is the enemy of security". This is certainly the case when security rules are created manually. In a complex network, security rules are derived from many different inputs: For example the corporate security policy may prohibit the use of peer to peer (P2P) applications; a department may have specific rules pertaining to its resources; and human resources may apply certain access rules for specific user classes. In the overall network all these rules have to be combined, including the resolution of conflicts for contradicting rules. Also in this case, abstraction to classes (of users, clients, servers, etc) helps in managing complexity.

6. FUTURE WORK

All the work presented here is qualitative, and therefore of limited use to real networks. While there are quantitative complexity measures for some subsystems of a network (graph theory, software complexity), a quantitative metric for overall network complexity, or at least for larger components would be very useful. Existing research on network modeling might provide a good starting point to provide a quantitative measure of complexity.

We stated that the rate of change differs, probably by an order of magnitude between the different categories of the model. Hardware may be changed significantly less frequently that configuration, which again changes less frequently than network state tables, such as the routing table. More work is required to understand the effect of potential coupling effects (e.g., between configuration and state tables) on the overall system complexity.

7. ACKNOWLEDGMENTS

The author would like to acknowledge Fred Baker, David Meyer and Bruno Klauser for many discussions, help and suggestions.

8. REFERENCES

[Biggs]: N. Biggs, "Algebraic graph theory", Cambridge University Press, London, 1974, Cambridge Tracts in Mathematics, No. 67.

[Bush]: R. Bush: "Complexity - The Internet and the Telco Philosophies"; 2002; http://www.nanog.org/meetings/nanog26/presentations/bushcomplex.pdf

[CIDR-Report]: The CIDR Report; www.cidr-report.org

[CIM]: The Common Information Model (CIM) http://www.dmtf.org/standards/cim/

[Constantine]: G. Constantine, "Graph complexity and the Laplacian matrix in blocked experiments", Linear and Multilinear Algebra 28 (1990), no. 1-2, 49-56.

[Doyle]: J. Doyle: "Networks and robustness" : http://www.cds.caltech.edu/~doyle/CmplxNets/

[Grone]: R. Grone and R. Merris, "A bound for the complexity of a simple graph", Discrete Math. 69 (1988), no. 1, 97-99.

[Kelly]: F. P. Kelly:"Modelling communication networks, present and future"; Phil. Trans. Roy. Soc. Lond. A354 (1996) 437-463

[McCabe]: T.J. McCabe: "A Complexity Measure"; IEEE Transactions on Software Engineering, December 1976

[Meyer]: D. Meyer: "Some Light Reading on Complexity and the Internet"

http://www.maoz.com/~dmm/complexity_and_the_internet/

[Neel]: "The Linear Complexity of a Graph"; David L. Neel, Michael E. Orrison; 2006; The electronic journal of combinatorics

[Quoitin]: B. Quoitin: "Modeling the Routing of an ISP with C-BGP"; NANOG 40 meetings archive; 2007

[RFC2622]: C. Alaettinoglu: "Routing Policy Specification Language (RPSL)"; June 1999

[RFC3439]: R. Bush; D. Meyer: "Some Internet Architectural Guidelines and Philosophy"; December 2002

[RFC4741]: R. Enns: "NETCONF Configuration Protocol"; December 2006

[Sangal]: N. Sangal et al: "Using Dependency Models to Manage Complex Software Architecture"; Proceedings OOPSLA 2005.
Also: http://www.eclipsezone.com/articles/lattix-dsm/

[Strassner]: J. Strassner: "Directory Enabled Networks"; ISBN 1578701406; 1999