Network Working Group
Internet-Draft
Intended status: Informational
Expires: October 27, 2016

M. Behringer
A. Retana
Cisco Systems
R. White
Ericsson
G. Huston
APNIC

April 25, 2016

A Framework for Defining Network Complexity
draft-behringer-ncrg-complexity-framework-02

Abstract

Complexity is a widely used parameter in network design, yet there is no generally accepted definition of the term. Complexity metrics exist in a wide range of research papers, but most of these address only a particular aspect of a network, for example the complexity of a graph or software. While it may be impossible to define a metric for overall network complexity, there is a desire to better understand the complexity of a network as a whole, as deployed today to provide Internet services. This document provides a framework to guide research on the topic of network complexity, as well as some practical examples for trade-offs in networking.

This document summarizes the work of the IRTF's Network Complexity Research Group (NCRG) at the time of its closure. It does not present final results, but a snapshot of an ongoing activity, as a basis for future work.

Table of Contents

1. Introduction

Network design can be described as the art of finding the simplest solution to solve a given problem. Complexity is thus assumed in the design process; engineers do not ask, "should there be complexity here," but rather, "how much complexity is required to solve this problem." This question, "how much complexity," assumes there is some way to characterize the amount of complexity present in a system. The reality is, however, this is an area of research and experience, rather than a solved problem within the network engineering space. Today's design decisions are made based on a rough estimation of the network's complexity, rather than a solid understanding.

The document begins with general considerations, including some foundational definitions and concepts. It then provides some examples for trade-offs that network engineers regularly make when designing a network. This section serves to demonstrate that there is no single answer to complexity; rather it is a managed trade-off between many parameters. After this, this document provides a set of parameters engineers should consider when attempting to either measure complexity or build a framework around it. This list makes no claim to be complete, but it serves as a guide of known existing areas of investigation, as well as a pointer to areas that still need to be investigated.

Two purposes are served here. The first is to guide researchers working in the area of complexity in their work. The more researchers are able to connect their work to the concerns of network designers, the more useful their research will become. This document may also guide research into areas not considered before. The second is to help network engineers to build a better understanding of where complexity might be "hiding" in their networks, and to be more fully aware of how complexity interacts with design and deployment.

The goal of the IRTF Network Complexity Research Group (NCRG) [ncrg] was to define a framework for network complexity research, while recognising that it may be impossible to define metrics for overall

network complexity. This document summarizes the work of this group at the time of its closure in 2014. It does not present final results, but a snapshot of an ongoing activity, as a basis for future work.

Many references to existing research in the area of network complexity are listed on the Network Complexity Wiki [wiki]. This wiki also contains background information on previous meetings on the subject, previous research, etc.

2. General Considerations

2.1. The Behavior of a Complex Network

While there is no generally accepted definition of network complexity, there is some understanding of the behavior of a complex network. It has some or all of the following properties:

- Self-Organization: A network runs some protocols and processes without external control; for example a routing process, failover mechanisms, etc. The interaction of those mechanisms can lead to a complex behaviour.

- Un-predictability: In a complex network, the effect of a local change on the behaviour of the global network may be unpredictable.

- Emergence: The behaviour of the system as a whole is not reflected in the behaviour of any individual component of the system.

- Non-linearity: An input into the network produces a non-linear result.

- Fragility: A small local input can break the entire system.

2.2. Complex versus Complicated

The two terms "complex" and "complicted" are often used interchangably, yet they describe different but overlapping properties. The RG made the following statements about the two terms, but they would need further refinement to be considered formal definitions:

- A "complicated" system is a deterministic system that can be understood by an appropriate level of analysis. It is often an externally applied attribute rather than an intrinsic property of a system, and is typically associated with systems that require deep or significant levels of analysis.

- A "complex" system, by comparison, is an intrinsic property of a system, and is typically associated with emergent behaviours, such that the behaviour of the system is not fully described by the sum of the behaviour of each of the components of the system. Complex systems are often associated with systems whose components exhibit high levels of interaction and feedback.

2.3. Robust Yet Fragile

Networks typically follow the "robust yet fragile" paradigm: They are designed to be robust against a set of failures, yet they are very vulnerable to other failures. Doyle [Doyle] explains the concept with an example: The Internet is robust against single component failure, but fragile to targeted attacks. The "robust yet fragile" property also touches on the fact that all network designs are necessarily making trade-offs between different design goals. The simplest one is articulated in "The Twelve Networking Truths" RFC1925 [RFC1925]: "Good, Fast, Cheap: Pick any two (you can't have all three)." In real network design, trade-offs between many aspects have to be made, including, for example, issues of scope, time and cost in the network cycle of planning, design, implementation and management of a network platform. Parameters are discussed in Section 4, and Section 3 gives some examples of tradeoffs.

2.4. The Complexity Cube

Complex tasks on a network can be done in different components of the network. For example, routing can be controlled by central algorithms, and the result distributed (e.g., OpenFlow model); the routing algorithm can also run completely distributed (e.g., routing protocols such as OSPF or ISIS), or a human operator could calculate routing tables and statically configure routing. Behringer [Behringer] defines these three axes of complexity as a "complexity cube" with three axes: Network elements, central systems, and human operators. Any function can be implemented in any of these three axes, and this choice likely has an impact on the overall complexity of the system.

2.5. Related Concepts

When discussing network complexity, a large number of influencing factors have to be taken into account to arrive at a full picture, for example:

- State in the network: Contains the network elements, such as routers, switches (with their OS, including protocols), lines, central systems, etc. The number and algorithmic complexity of the protocols on network devices for example.

- Human operators: Complexity manifests itself often by a network that is not completely understood by human operators. Human error is a primary source for catastrophic failures, and therefore must be taken into account.

- Classes / templates: Rather than counting the number of lines in a configuration, or the number of hardware elements, more important is the number of classes from which those can be derived. In other words, it is probably less complex to have 1000 interfaces which are identically configured than 5 that are completely different configured.

- Dependencies and interactions: The number of dependencies between elements, as well as the interactions between them has influence on the complexity of the network.

- TCO (Total cost of ownership): TCO could be a good metric for network complexity, if the TCO calculation takes into account all influencing factors, for example training time for staff to be able to maintain a network.

- Benchmark Unit Cost is a related metric that indicates the cost of operating a certain component. If calculated well, it reflects at least parts of the complexity of this component. Therefore, the way TCO or BUC are calculated can help to derive a complexity metric.

- Churn / rate of change: The change rate in a network itself can contribute to complexity, especially if a number of components of the overall network interact.

Networks differ in terms of their intended purpose (such as is found in differences between enterprise and public carriage network platforms, and in their intended role (such as is found in the differences between so-called "access" networks and "core" transit networks). The differences in terms of role and purpose can often lead to differences in the tolerance for, and even the metrics of, complexity within such different network scenarios. This is not necessarily a space where a single methodology for measuring complexity, and defining a single threshold value of acceptability of complexity, is appropriate.

2.6. Technical Debt

Many changes in a network are made with a dependency on the existing network. Often, a suboptimal decision is made because the optimal decision is hard or impossible to realise at the time. Over time, the number of suboptimal changes in themselves cause significant

complexity, which would not have been there had the optimal solution been implemented.

The term "technical debt" refers to the accumulated complexity of sub-optimal changes over time. As with financial debt, the idea is that also technical debt must be repaid one day by cleaning up the network or software.

2.7. Layering considerations

In considering the larger space of applications, transport services, network services and media services, it is feasible to engineer responses for certain types of desired applications responses in many different ways, and involving different layers of the so-called network protocol stack. For example, Quality of Service could be engineered at any of these layers, or even in a number of combinations of different layers.

Considerations of complexity arise when mutually incompatible measures are used in combination (such as error detection and retransmission at the media layer in conjunction with the use TCP transport protocol), or when assumptions used in one layer are violated by another layer. This results in surprising outcomes that may result in complex interactions, for example oscillation because different layers use different timers for retransmission. These issues have led to the perspective that increased layering frequently increases complexity [RFC3439].

While this research work is focussed network complexity, the interactions of the network with the end-to-end transport protocols, application layer protocols and media properties are relevant considerations here.

3. Tradeoffs

Network complexity is a system level, rather than component level, problem; overall system complexity may be more than the sum of the complexity of the individual pieces.

There are two basic ways in which system level problems might be addressed: interfaces and continuums. In addressing a system level problem through interfaces, we seek to treat each piece of the system as a "black box," and develop a complete understanding of the interfaces between these black boxes. In addressing a system level problem as a continuum, we seek to understand the impact of a single change or element to the entire system as a set of tradeoffs.

While network complexity can profitably be approached from either of these perspectives, in this document we have chosen to approach the system level impact of network complexity from the perspective of continuums of tradeoffs. In theory, modifying the network to resolve one particular problem (or class of problems) will add complexity which results in the increased likelihood (or appearance) of another class of problems. Discovering these continuums of tradeoffs, and then determining how to measure each one, become the key steps in understanding and measuring system level complexity in this view.

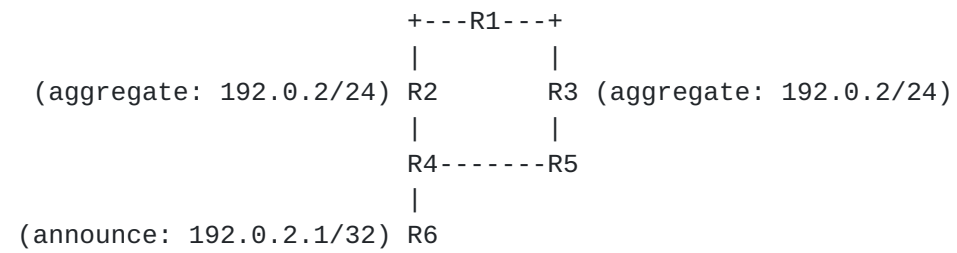
The following sections describe five such continuums; more may be possible.

- Control Plane State versus Optimal Forwarding Paths (or its opposite measure, stretch)
- Configuration State versus Failure Domain Separation
- Policy Centralization versus Optimal Policy Application
- Configuration State versus Per Hop Forwarding Optimization
- Reactivity versus Stability

3.1. Control Plane State versus Optimal Forwarding Paths (Stretch)

Control plane state is the aggregate amount of information carried by the control plane through the network in order to produce the forwarding table at each device. Each additional piece of information added to the control plane --such as more specific reachability information, policy information, additional control planes for virtualization and tunneling, or more precise topology information-- adds to the complexity of the control plane. This added complexity, in turn, adds to the burden of monitoring, understanding, troubleshooting, and managing the network.

Removing control plane state, however, is not always a net positive gain for the network as a system; removing control plane state almost always results in decreased optimality in the forwarding and handing of packets travelling through the network. This decreased optimality can be termed stretch, which is defined as the difference between the absolute shortest (or best) path traffic could take through the network and the path the traffic actually takes. Stretch is expressed as the difference between the optimal and actual path. The figure below provides and example of this tradeoff.



Assume each link is of equal cost in this figure, and R6 is advertising 192.0.2.1/32.

For R1, the shortest path to 192.0.2.1/32, advertised by R6, is along the path [R1,R2,R4,R6].

Assume, however, the network administrator decides to aggregate reachability information at R2 and R3, advertising 192.0.2.0/24 towards R1 from both of these points. This reduces the overall complexity of the control plane by reducing the amount of information carried past these two routers (at R1 only in this case).

Aggregating reachability information at R2 and R3, however, may have the impact of making both routes towards 192.0.2.1/32 appear as equal cost paths to R1; there is no particular reason R1 should choose the shortest path through R2 over the longer path through R3. This, in effect, increases the stretch of the network. The shortest path from R1 to R6 is 3 hops, a path that will always be chosen before aggregation is configured. Assuming half of the traffic will be forwarded along the path through R2 (3 hops), and half through R3 (4 hops), the network is stretched by ((3+4)/2) - 3), or .5, a "half a hop."

Traffic engineering through various tunneling mechanisms is, at a broad level, adding control plane state to provide more optimal forwarding (or network utlization). Optimizing network utilization may require detuning stretch (intentionally increasing stretch) to increase overall network utilization and efficiency; this is simply an alternate instance of control plane state (and hence complexity) weighed against optimal forwarding through the network.

3.2. Configuration State versus Failure Domain Separation

A failure domain, within the context of a network control plane, can be defined as the set of devices impacted by a change in the network topology or configuration. A network with larger failure domains is more prone to cascading failures, so smaller failure domains are normally preferred over larger ones.

The primary means used to limit the size of a failure domain within a network's control plane is information hiding; the two primary types of information hidden in a network control plane are reachability information and topology information. An example of aggregating reachability information is summarizing the routes 192.0.2.1/32, 192.0.2.2/32, and 192.0.2.3/32 into the single route 192.0.2.0/24, along with the aggregation of the metric information associated with each of the component routes. Note that aggregation is a "natural" part of IP networks, starting with the aggregation of individual hosts into a subnet at the network edge. An example of topology aggregation is the summarization of routes at a link state flooding domain boundary, or the lack of topology information in a distance-vector protocol.

While limiting the size of failure domains appears to be an absolute good in terms of network complexity, there is a definite tradeoff in configuration complexity. The more failure domain edges created in a network, the more complex configuration will become. This is particularly true if redistribution of routing information between multiple control plane processes is used to create failure domain boundaries; moving between different types of control planes causes a loss of the consistent metrics most control planes rely on to build loop free paths. Redistribution, in particular, opens the door to very destructive positive feedback loops within the control plane. Examples of control plane complexity caused by the creation of failure domain boundaries include route filters, routing aggregation configuration, and metric modifications to engineer traffic across failure domain boundaries.

Returning to the network described in the previous section, aggregating routing information at R2 and R3 will divide the network into two failure domains: (R1,R2,R3), and (R2,R3,R4,R5). A failure at R5 should have no impact on the forwarding information at R1.

A false failure domain separation occurs, however, when the metric of the aggregate route advertised by R2 and R3 is dependent on one of the routes within the aggregate. For instance, if the metric of the 192.0.2.0/24 aggregate is derived from the metric of the component 192.0.2.1/32, then a failure of this one component will cause changes in the forwarding table at R1 --in this case, the control plane has not truly been separated into two distinct failure domains. The added complexity in the illustration network would be the management of the configuration required to aggregate the contorl plane information, and the management of the metrics to ensure the control plane is truly separated into two distinct failure domains.

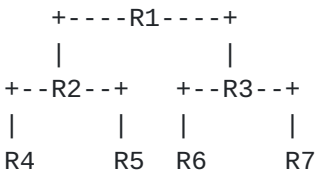
Replacing aggregation with redistribution adds the complexity of managing the feedback of routing information redistributed between

the failure domains. For instance, if R1, R2, and R3 were configured to run one routing protocol, while R2, R3, R4, R5, and R6 were configured to run another protocol, R2 and R3 could be configured to redistribute reachability information between these two control planes. This can split the control plane into multiple failure domains (depending on how, specifically, redistribution is configured), but at the cost of creating and managing the redistribution configuration. Futher, R3 must be configured to block routing information redistributed at R2 towards R1 from being redistributined (again) towards R4 and R5.

3.3. Policy Centralization versus Optimal Policy Application

Another broad area where control plane complexity interacts with optimal network utilization is Quality of Service (QoS). Two specific actions are required to optimize the flow of traffic through a network: marking and Per Hop Behaviors (PHBs). Rather than examining each packet at each forwarding device in a network, packets are often marked, or classified, in some way (typically through Type of Service bits) so they can be handled consistently at all forwarding devices.

Packet marking policies must be configured on specific forwarding devices throughout the network. Distributing marking closer to the edge of the network necessarily means configuring and managing more devices, but produces optimal forwarding at a larger number of network devices. Moving marking towards the network core means packets are marked for proper handling across a smaller number of devices. In the same way, each device through which a packet passes with the correct PHBs configured represents an increase in the consistency in packet handling through the network as well as an increase in the number of devices which must be configured and managed for the correct PHBs. The network below is used for an illustration of this concept.



In this network, marking and PHB configuration may be configured on any device, R1 through R7.

Assume marking is configured at the network edge; in this case, four devices, (R4,R5,R6,R7), must be configured, including ongoing configuration management, to mark packets. Moving packet marking to R2 and R3 will halve the number of devices on which packet marking

configuration must be managed, but at the cost of inconsistent packet handling at the inbound interfaces of R2 and R3 themselves.

Thus reducing the number of devices which must have managed configurations for packet marking will reduce optimal packet flow through the network. Assuming packet marking is actually configured along the edge of this network, configuring PHBs on different devices has this same tradeoff of managed configuration versus optimal traffic flow. If the correct PHBs are configured on R1, R2, and R3, then packets passing through the network will be handled correctly at each hop. The cost involved will be the management of PHB configuration on three devices. Configuring a single device for the correct PHBs (R1, for instance), will decrease the amount of configuration management required, at the cost of less than optimal packet handling along the entire path.

3.4. Configuration State versus Per Hop Forwarding Optimization

The number of PHBs configured along a forwarding path exhibits the same complexity versus optimality tradeoff described in the section above. The more classes (or queues) traffic is divided into, the more fine-grained traffic will be managed as it passes through the network. At the same time, each class of service must be managed, both in terms of configuration and in its interaction with other classes of service configured in the network.

3.5. Reactivity versus Stability

The speed at which the network's control plane can react to a change in configuration or topology is an area of widespread study. Control plane convergence can be broken down into four essential parts:

- Detecting the change
- Propagating information about the change
- Determining the best path(s) through the network after the change
- Changing the forwarding path at each network element along the modified paths

Each of these areas can be addressed in an effort to improve network convergence speeds; some of these improvements come at the cost of increased complexity.

Changes in network topology can be detected much more quickly through faster echo (or hello) mechanisms, lower layer physical detection, and other methods. Each of these mechanisms, however, can only be

used at the cost of evaluating and managing false positives and high rates of topology change.

If the state of a link change can be detected in 10ms, for instance, the link could theoretically change state 50 times in a second --it would be impossible to tune a network control plane to react to topology changes at this rate. Injecting topology change information into the control plane at this rate can destabalize the control plane, and hence the network itself. To counter this, most fast down detection techniques include some form of dampening mechanism; configuring and managing these dampening mechanisms increases complexity that must be configured and managed.

Changes in network topology must also be propagated throughout the network, so each device along the path can compute new forwarding tables. In high speed network environments, propagation of routing information changes can take place in tens of milliseconds, opening the possibility of multiple changes being propagated per second. Injecting information at this rate into the contral plane creates the risk of overloading the processes and devices participating in the control plane, as well as creating destructive positive feedback loops in the network. To avoid these consequences, most control plane protocols regulate the speed at which information about network changes can be transmitted by any individual device. A recent innovation in this area is using exponential backoff techniques to manage the rate at which information is advertised into the control plane; the first change is transmitted quickly, while subsequent changes are transmitted more slowly. These techniques all control the destabalilzing effects of rapid information flows through the control plane through the added complexity of configuring and managing the rate at which the control plane can propagate information about network changes.

All control planes require some form of algorithmic calculation to find the best path through the network to any given destination. These algorithms are often lightweight, but they still require some amount of memory and computational power to execute. Rapid changes in the network can overwhelm the devices on which these algorithms run, particularly if changes are presented more quickly than the algorithm can run. Once the devices running these algorithms become processor or memory bound, it could experience a computational failure altogether, causing a more general network outage. To prevent computational overloading, control plane protocols are designed with timers limiting how often they can compute the best path through a network; often these timers are exponential in nature, allowing the first computation to run quickly, while delaying subsequent computations. Configuring and managing these timers is another source of complexity within the network.

Another option to improve the speed at which the control plane reacts to changes in the network is to precompute alternate paths at each device, and possibly preinstall forwarding information into local forwarding tables. Additional state is often needed to precompute alternate paths, and additional algorithms and techniques are often configured and deployed. This additional state, and these additional algorithms, add some amount of complexity to the configuration and management of the network.

In some situations (for some topologies), a tunnel is required to pass traffic around a network failure or topology change. These tunnels, while not manually configured, represent additional complexity at the forwarding and control planes.

4. Parameters

In Section 3 we describe a set of trade-offs in network design to illustrate the practical choices network operators have to make. The amount of parameters to consider in such tradeoff scenarios is very large, thus that a complete listing may not be possible. Also the dependencies between the various metrics itself is very complex and requires further study. This document attempts to define a methodology and an overall high level structure.

To analyse tradeoffs it is necessary to formalise them. The list of parameters for such tradeoffs is long, and the parameters can be complex in themselves. For example, "cost" can be a simple unidimensional metric, but "extensibility" or "optimal forwarding state" are harder to define in detail.

A list of parameters to trade off contains metrics such as:

- State: How much state needs to be held in control plane, forwarding plane, configuration, etc.
- Cost: How much does the network cost to build (capex) and run (opex)
- Bandwidth / delay / jitter: Traffic characteristics between two points (average, max, ...)
- Configuration complexity: How hard to configure and maintain the configuration
- Susceptibility to Denial-of-Service: How easy is it to attack the service

- Security (confidentiality / integrity): How easy is it to sniff / modify / insert the data flow
- Scalability: To what size can I grow the network / service
- Stability: How stable is the network under the influence of local change?
- Reactivity: How fast does the network converge, or adapt to new situations?
- Extensibility: Can I use the network for other services in the future?
- Ease of troubleshooting: Are failure domains separated? How hard is it to find and correct problems?
- Optimal per-hop forwarding behavior
- Predictability: If I change a parameter, what will happen?
- Clean failure: When a problem arises, does the root cause lead to deterministic failure

5. Elements of Complexity

Complexity can be found in various elements in a networked system. For example, the configuration of a network element reflects some of the complexity contained in this system. Or an algorithm used by a protocol may be more or less complex. When classifying complexity the first question to ask is "WHAT is complex?". This section offers a method to answer this question.

5.1. The Physical Network (Hardware)

The set of network devices and wiring contains a certain complexity. For example, adding a redundant link between two locations increases the complexity of the network, but provides more redundancy. Also network devices can be more or less modular, which has impact on complexity trading off against ease of maintenance, availability and upgradability.

5.2. Algorithms

The behavior of the physical network is not only defined by the hardware, but also by algorithms that run on network elements and in central locations. Every algorithm has a certain intrinsic complexity, which is the subject of research on software complexity.

5.3. State in the Network

The way a network element treats traffic is defined largely by the state in the network, in form of configuration, routing state, security measures, etc. Section 3.1 shows an example where more control plane state allows a more precise forwarding.

5.4. Churn

The rate of change itself is a parameter in complexity, which needs to be weighed against other parameters. Section 3.5 explains a trade-off between the speed of communicating changes through the network and the stability of the network.

5.5. Knowledge

Certain complexity parameters have a strong link to the human aspect of networking. For example, the more option and parameters a network protocol has, the harder it is to configure and trouble-shoot. Therefore, there is a trade-off between the knowledge to be maintained by operational staff and desired functionality. The required knowledge of network operators is therefore an important part in complexity considerations.

6. Location of Complexity

The previous section discussed in which form complexity may be perceived. This section focuses on where this complexity is located in a network. For example, an algorithm can run centrally, distributed, or even in the head of a network administrator. In classifying the complexity of a network, the location of a component may have an impact on overall complexity. This section offers a methodology to the question "WHERE is the complex component?"

6.1. Topological Location

An algorithm can run distributed, for example a routing protocol like OSPF runs on all routers in a network. But it can also be in a central location such as the Network Operations Center (NOC). The physical location has impact on several other parameters, such as availability (local changes might be faster than going through a remote NOC) and ease of operation, because it might be easier to understand and troubleshoot one central entity rather than many remote ones.

The example in Section 3.3 shows how the location of state (in this case configuration) impacts the precision of the policy enforcement

and the corresponding state required. Enforcement closer to the edge requires more network wide state, but is more precise.

6.2. Logical Location

Independent of its physical location, the logical location also may make a difference to complexity. A controller function for example can reside in a NOC, but also on a network element. Generally, organising a network in separate logical entities is considered positive, because it eases the understanding of the network, thereby making trouble-shooting and configuration easier. For example a BGP route reflector is a separate logical entity from a BGP speaker, but it may reside on the same physical node.

6.3. Layering Considerations

Also the layer of the TCP/IP stack in which a function is implemented can have an impact on the complexity of the overall network. Some functions are implemented in several layers in slightly different ways, which may lead to unexpected results.

As an example, a link failure is detected on various layers: L1, L2, the IGP, BGP, and potentially more. Since those have dependencies on each other, different link failure detection times can cause undesired effects. Dependencies are discussed in more detail in the next section.

7. Dependencies

Dependencies are generally regarded as related to overall complexity. A system with less dependencies is generally considered less complex. This section proposes a way to analyse dependencies in a network.

For example, [Chun] states: "We conjecture that the complexity particular to networked systems arises from the need to ensure state is kept in sync with its distributed dependencies."

In this document we distinguish three types of dependencies: Local dependencies, network wide dependencies, and network external dependencies.

7.1. Local Dependencies

Local dependencies are relative to a single node in the network. For example, an interface on a node may have an IP address; this address may be used in other parts of the configuration. If the interface address changes, the dependent configuration parts have to change as well.

Similar dependencies exist for QoS policies, access-control-lists, names and numbers of configuration parts, etc.

7.2. Network Wide Dependencies

Routing protocols, failover protocols, and many other have dependencies across the network. If one node is affected by a problem, this may have a ripple effect through the network. These protocols are typically designed to deal with unexpected consequences, thus unlikely to cause an issue on their own. But occasionally a number of complexity issues come together, for example, different timers on different layers, then unexpected behaviour can occur.

7.3. Network External Dependencies

Some dependencies are on elements outside the actual network, for example on an external NTP clock source, or a AAA server. Again, a tradeoff is made: In the example of AAA used for login authentication, we reduce the configuration (state) on each node, specifically user specific configuration. But we add an external dependency on a AAA server. In networks with many administrators, a AAA server is clearly the only manageable way to track all administrators. But it comes at the cost of this external dependency, with the consequence that admin access may be lost for all devices at the same time, when the AAA server is unavailable.

Even with the external dependency on a AAA server, the advantage of centralizing the user information (and logging) still has significant value over distributing user information across all devices. To solve the problem of the central dependency not being available, other solutions have been developed, for example a secondary authentication mode with a single root level password in case the AAA server is not available.

8. Management Interactions

A static network generally is relatively stable; conversely, changes introduce a degree of uncertainty and therefore need to be examined in detail. Also, the trouble shooting of a network exposes intuitively the complexity of the network. This section proposes a methodology to classify management interactions with regard to their relationship to network complexity.

8.1. Configuration Complexity

Configuration can be seen as distributed state across network devices, where the administrator has direct influence on the operation of the network. Modifying the configuration can improve the network behaviour over all, or negatively affect it. In the worst case, a single misconfiguration could potentially bring down the entire network. Therefore it is important that a human administrator can manage the complexity of the configuration well.

The configuration reflects most of the local and global dependencies in the network, as explained in Section 7. Tracking those dependencies in the configuration helps in understanding the overall network complexity.

8.2. Troubleshooting Complexity

Unexpected behaviour can have a number of sources: The configuration may contain errors, the operating system (algorithms) may have bugs, and the hardware may be faulty, which includes anything from broken fibres to faulty line cards. In serious problems, a combination of causes could result in a single visible condition. Tracking the root causes of a error condition may be extremely difficult, pointing to the complex nature of a network.

Being able to find the source of a problem requires therefore a solid understanding of the complexity of a network. The configuration complexity discussed in the previous section represents only a part of the overall problem space.

8.3. Monitoring Complexity

Even in the absence of error conditions, the state of the network should be monitored to detect error conditions ideally before network services are affected. For example, a single "link-down" event may not cause a service disruption in a well designed network, but the problem needs to be resolved quickly to restore redundancy.

Monitoring a network has itself a certain complexity. Issues are in scale, variations of devices to be monitored, variations of methods used to collect information, the inevitable loss of information as reporting is aggregated centrally, and the knowledge required to understand the network, the dependencies and interactions with users and other external inputs.

8.4. Complexity of System Integration

A network doesn't just consist of network devices, but includes a vast array of backend and support systems. It also interfaces a large variety of user devices, and a number of human interfaces, both to the user / customer, as well as to administrators of the network. To make sure the overall network provides the overall service expected requires a system integration job.

All those interactions and systems have to be modelled to understand the inter-dependencies and complexities in the network. This is a large area of future research.

9. External Interactions

A network is not a self-contained entity, but exists to provide connectivity and services to users and other networks, both of which are outside the direct control of a network administrator. The user experience of a network also illustrates a form of interaction with its own complexity.

External interactions fall into the following categories:

User Interactions: Users need a way to request a service, to have their problems resolved, and potentially to get billed for their usage. There are a number of human interfaces that need to be considered, which depend to some extend on the network, for example for troubleshooting, or monitoring usage.

Interactions with End Systems: The network also interacts with the devices that connect to it. Typically a device receives an IP address from the network, and information on how to resolve domain names, plus potentially other services. While those interactions are relatively simple, the vast amount of end device types makes this a complicated space to track.

Inter-Network Interactions: Most networks connect to other networks. Also in this case there are many interactions between networks, both technically (for example, running a routing protocol), as well as non-technical (for example, tracing problems across network boundaries).

For a fully operational network providing services to users, also the external interactions and dependencies form an integral part of the overall complexity of the network service. A specific example are the root DNS servers, which are critical to the function of the Internet. Practically all Internet users have an implicit dependency on the root DNS servers, which explains why those are frequent

targets for attacks. Understanding the overall complexity of a network includes understanding all those external dependencies. Of course, in the case of the root DNS servers, there is little a network operator can influence.

10. Examples

In the foreseeable future it is unlikely to define a single, objective metric that includes all the relevant aspects of complexity. In the absence of such a global metric, a comparative approach could be easier.

For example, it is possible to compare the complexity of a centralised systems where algorithms run centrally, and the results are distributed to the network nodes with a distributed algorithm. The type of algorithm may be similar, but the location is different, and a different dependency graph would result. The supporting hardware may be the same, thus could be ignored for this exercise. Also layering is likely to be the same. The management interactions though would significantly differ in both cases.

The classification in this document also makes it easier to survey existing research with regards to which area of complexity is covered. This could help in identifying open areas for research.

11. Security Considerations

This document does not discuss any specific security considerations.

12. Acknowledgements

The motivations and framework of this overview of studies into network complexity is the result of many meetings and discussions, with too many people to provide a full list here. However, key contributions have been made by: John Doyle, Dave Meyer, Jon Crowcroft, Mark Handley, Fred Baker, Paul Vixie, Lars Eggert, Bob Briscoe, Keith Jones, Bruno Klauser, Steve Youell, Joel Obstfeld, Philip Eardley.

The authors would like to acknowledge the contributions of Rana Sircar, Ken Carlberg and Luca Caviglione in the preparation of this document.

13. Informative References

[Behringer]
Behringer, M., "Classifying Network Complexity", Proceedings of the ACM Re-Arch'09, December 2009.

[Chun] Chun, B-G., Ratnasamy, S., and E. Eddie, "NetComplex: A Complexity Metric for Networked System Design", 5th Usenix Symposium on Networked Systems Design and Implementation NSDI 2008, April 2008, <http://berkeley.intel-research.net/sylvia/netcomp.pdf>.

[Doyle] Doyle, J., "The 'robust yet fragile' nature of the Internet", PNAS vol. 102 no. 41 14497-14502, October 2005.

[ncrg] "Network Complexity Research Group", <https://irtf.org/concluded/ncrg>.

[RFC1925] Callon, R., "The Twelve Networking Truths", RFC 1925, DOI 10.17487/RFC1925, April 1996, <http://www.rfc-editor.org/info/rfc1925>.

[RFC3439] Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, DOI 10.17487/RFC3439, December 2002, <http://www.rfc-editor.org/info/rfc3439>.

[wiki] "Network Complexity Wiki", <http://networkcomplexity.org/>.

Authors' Addresses

Michael H. Behringer
Cisco Systems
Building D, 45 Allee des Ormes
Mougins 06250
France

Email: mbehring@cisco.com

Alvaro Retana
Cisco Systems
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Email: aretana@cisco.com

Russ White
Ericsson
144 Warm Wood Lane
Apex, NC 27539
United States

Email: russw@riw.us
URI: http://www.ericsson.com

Geoff Huston
Asia Pacific Network Information Centre
6 Cordelia St
South Brisbane, QLD 4101
Australia

Email: gih@apnic.net
URI: http://www.apnic.net