The New Telecom Ecosystem:

Complex Systems & Autonomic Networks

Wedge Greene

LTC International

Date: 07/06/2006

Article originally appeared as blog entries in InsideOut
http://www.ltcinternational.com/inside-out/

Expect Results®

THE NEW TELECOM ECOSYSTEM:

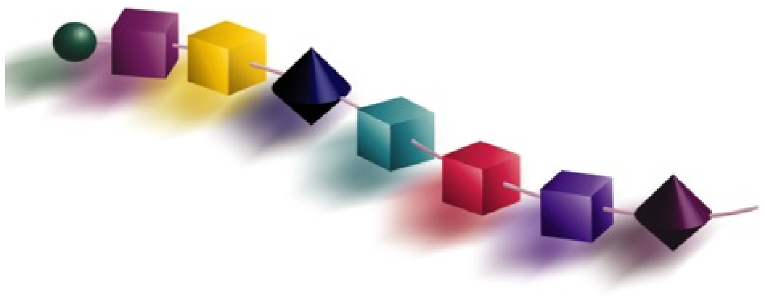
# 2 - COMPLEX SYSTEMS & AUTONOMIC NETWORKS

Summary:

Collected from Blog entries of Wedge Greene in the summer of 2006 (read first "#1 - Preparing for the New Ecosystem").

As an aid in helping the reader reach the same paradigm changing conclusions that I reached in the late '90's, I give a personal history that lead to these new approaches. I was not alone in this journey and participate in a community which was collectively exploring this new territory. Many minds came to the conclusion that we were viewing networks wrong, and in this exploration, I particularly saw this had a profound impact on how network management should occur. The companion first half of this collection of themed blogs from the summer of 2006, described the development of NewWave OSS and the start of the NGOSS program – our first attempt at a solution. But NGOSS, as my team conceived it, was always to be an interim step to an even more advanced management paradigm…

Business Operations Architects ®



Part 1: Biography – intellectual genesis of NewWave OSS

"The dominant approach to regulating communications networks treats each network component as if it existed in isolation. In so doing, the current approach fails to capture one of the essential characteristics of networks, which is the complex manner in which components interact with one another when combined into an integrated system."
- Daniel F. Spulber & Christopher S. Yoo

Learning not "to see out of the box" but to **grok** "the box and its environment"

This article is in the first person as was the original blog entries: hopefully by showing the personal route I took, it will be easier to understand the conclusions I reached.

Today there is a name for what I studied in college in the 70's and cobbled together for a special project major: **Complex Systems**. I did not myself understand how 'what I was studding' came together, but I knew it was organized and systematic. Luckily, Herb York, ex advisor to 4 presidents and instigator of ARPA and NASA, let me slide in under his 'fuzzy' institute – else I would likely never have graduated. Back in the early 1970's, there wasn't even a computer science department at UCSD; so of course Complex Systems did not have a name, department or a major. I had to jump around collages to keep a program of study going.

But from my first solo college computer programming project in 1970, a nonlinear oscillator simulation, I was hooked on complex systems. There is something elegant and continually surprising about complex systems that I did not find in traditional science; except as unresolved examples on the edge of theory and practice. So I jumped about; spent time in physics, lots of projects in population biology and evolution, theory of statistics, economics, work in resource modeling which lead to modeling social and political dynamics. All facilitated by the lovely *difference equation*.

For a long time I was determined to predicatively model the world or parts thereof, and was very frustrated by how hard it was to get our multivariate, nonlinear system models to converge. Then in graduate school I became a **Bayesian Utilitarian**. I suddenly understood that we had it all wrong about what the models were supposed to do – not to converge to a solution and predict future outcomes – but to *help us find the missing information and gain understanding about the system we were modeling*. Most of my doctorial committee supported me in this insight, but the university departmental program was quite antagonistic, the money dried up, so I gave up and dropped out. Of course, in later life, I used these skills when I could, for instance using graph theory to unwind & document complex COBOL programs, but mostly I was too frustrated to continue these studies.

The point of describing this diverse study is - that to understand a cow is a manual, and also is an animal; you have to study a lot of different animals. **To recognize and understand complex systems, you need to look at a lot of different system types in different fields of study.** Each technology zone creates complex systems with different characteristics and slightly different systems behavior. But complex systems also show similar properties and common behavior independent of the technology base. Personally, a lot of cognitive dissidence needed to build up before I could see this shift.

Building Networks

It also turns out that during my graduate studies in 1976-8, my doctorial advisor (Paul Werbos) got a DARPA grant, and his team got logged into the ARPAnet, sharing computing resources for our rather big simulations. For two decades following, while making a living as a software guy, I followed advances in networking technology as a hobby. I watched what some really smart people were doing within the IETF & with campus networks – but I had no wish to go anywhere near academic institutions again. During these years, commercial networking was exclusively SNA private line and x.25 – both pretty boring. While the IETF work in packet routing systems was intellectually engaging to follow, there seemed to be no connection with this and paying bills.

Than, in 1990, a few people in MCI, technically & inspirationally lead by David McDysan, proposed building a commercial data network. David recruited me to be part of the founding team of this "startup within a big company" and off we went. A great, wild ride for a decade: 4 networks conceived and built, growing from zero to $ 1 billion revenue. We deployed in succession Switched TDM, Frame Relay, ATM, commercial Internet, and then internet services like hosting and commerce. We began linking up networks with BT and all the Concert 2nd tier carriers. We had new developments or twists confront us every month - You should have been there!

Towards the beginning of the Frame Relay product introduction, because I was the software guy, I got shifted, pretty much kicking and screaming, from signaling and routing to OSS. Back then OSS, in the words of David Russell who forced the assignment on me, was considered "the back end of the donkey" and held in relative networking esteem similar to how a stable hand is in equestrian sports. David Russell, a heroic military mind, had once been DARPA lead for the founding of ARPAnet and hired this kid Vint Cerf to build TCP/IP – Dave was the 'Grandfather of the Internet' so attempting to doge this career redirection was pretty much futile. Lucky for me, he won.

Positively, the network management problems were real and it was fresh ground to be creative in systems design and scaling application systems. Mostly there were no vendors, so we had to build it ourselves or help vendors build it to our wishes. Along the way, as I got a handle on what OSS needed to do, I began, around 1996 to apply my old love of complex systems toward the understanding of data networks. *This was the intellectual geneses that lead to NewWave OSS and eventually my Fine Grain NGOSS proposal.*

Today, I am one of a very small number of telecom people who see communication networks as Complex Systems - Ironic, because the study of Complex Systems is now strong and scientifically popular. The tools and approaches available in Complex Systems today are impressive. There is untapped wealth here which will remake communications. But networks have not enjoyed the profound advances in complex systems science. It is understandable; skepticism is strong. Wikipedia Encyclopedia, today miss-defines Complex Systems this way:

> "Many natural phenomena can be considered to be **complex systems**, and their study (*complexity science*) is highly interdisciplinary… Beyond the fact that these things are all ***networks*** of some kind, and that they are complex, it may appear that they have little in common, hence that the term "complex system" is vacuous."
>
> - Wikipedia: Complex Systems

Yet, while saying complex systems all were networks (biological networks, social networks, etc.), Wikipedia does not list communications networks with the half dozen fields where Complex Systems can be applied.

While application of Complex Systems to telecommunications networks is still new and small, great potential exists – analogous to the gains which occurred when information theory made packet networks realistic. But making the shift from Classical OSS thinking, to seeing networks as Complex Systems, is not easy. It requires a paradigm shift – and paradigm shifts are rocky.

> “New assumptions (paradigms/theories) require the reconstruction of prior assumptions and the reevaluation of prior facts. This is difficult and time consuming. It is also strongly resisted by the established community. When a shift takes place, "a scientist's world is qualitatively transformed [and] quantitatively enriched by fundamental novelties of either fact or theory."
> - Thomas Kuhn: The Structure of Scientific Revolutions; recapped by Frank Pajares.

An analysis of Complex Systems applied to telecommunications is a first step toward designing autonomic networks and network control systems. An **Autonomic Network** *will independently take action to restore itself*. We extend the definition of the network complex system to include the network and the support structures (software and servers) of that network. We gave some strong reasons for why this is necessary in the earlier series “Toward a New Telecom Ecosystem” and we will continue to do so; but basically, it is impossible today to artificially divide and separate the network from the support systems, internet middleware, and products which are associated with the network – it functions *if and only if* all parts work in concert. In total, these form the complex system.

> At the start, a new candidate for paradigm may have few supporters (and the motives of the supporters may be suspect). If the supporters are competent, they will (a) improve the paradigm, (b) explore its possibilities, (c) and show what it would be like to belong to the community guided by it. For the paradigm destined to win, the number and strength of the persuasive arguments in its favor will increase.
> - Thomas Kuhn: The Structure of Scientific Revolutions; recapped by Frank Pajares.

In this colleted paper on ***Complexity & Autonomic Systems***, I will present an introduction of why this approach is needed now, and how complex systems can be applied to telecom. In parts 2 and 3 we will look at how incorrect expressions of products and network can cloud engineering designs and ultimately produce negative customer reactions. Part 4 will explore networks as stable changing systems. Part 5 will will look at **emergent behavior** in networks and OSS. To make this easier to see, lets use the familiar Frame Relay Billing example to illustrate the importance of seeing the network.

The next few parts are quite complex, and might need to be read several times. I apologize for this difficulty, but the challenge of following these arguments is an inherent part of the process of change to a better way of building OSS. This is about shifting the way we, as an industry, have always done things. In fact, the more simple the new idea and approach, the more elegant the outcome - hopefully, I will get help with this.

Note: I still use older examples, mostly from Frame Relay, because these networks, products, and supplier companies are generally history, so there are fewer people to offend by recounting these events.

Part 2: “The map is not the territory”

> “The word is not the thing … The map is not the territory ... The only usefulness of a map depends on similarity of structure between the empirical world and the map...
> - Alfred Korzybski, *Science and Sanity*:

“The father of general semantics, Alford Korzybski stated, "A map is not the territory it represents, but if correct, it has a similar structure to the territory, which accounts for its usefulness". What this means is that our perception of reality is not reality itself but our own version of it, or our "map". [- recapped from Korzybski by Rex Steven Sikes]

Deeper Look at Frame Relay Billing Example:

Remember our parallel test of database association and aggregator association (“# 1 – Preparing The New Telecom Ecosystem”, Part 6). A problem with both approaches was their *reliance on a virtual information model*; external meta-data to determine which endpoint of a frame relay access circuit matched which other endpoint to form the virtual circuit.

Probability of the data: This data needed to be verified as part of another process of inputting or verifying data. If this processes generated bad information, our derived billing record for a day’s usage of a frame relay circuit was also incorrect. So a better design would have linked “a process to verify the endpoints were the endpoints of the same circuit”, by going to the network for the data, at the same time as the usage information was collected. Best, some transactional method would be used to link the verification of endpoints and the collection of usage information, as occurring in the same small fraction of time and as atomically linked actions. When doing this, the aggregator design became “near-real-time” in binding information. Precision of the data was insured to a great degree.

But a great 18th century thinker Thomas Bayes once described a formula for reality: I paraphrase this as “The probability of the data being true is adjusted by the **probability of the model**.” A simple way of seeing the significance of this is that if the model is wrong, there is a very good chance the data associations made from the model are also wrong. It so happens that in the frame relay billing model above, the model is an abstraction of an external fabricated idea, and also application of the older “circuit” language, onto a newer network design and quite different network protocols - the model has no real, direct relationship to the network. This billing model was based on the older notion of a circuit (a continuous pipe for passing data information signals). Frame relay as a product was supplanting Private Line (T1 and T3 or E1) communications systems. In Private Line, the notion of a circuit is much closer to physical reality (but still not exactly correct). However this was not so in frame relay.

First of all, frame relay was a point-to-multiple-point addressing protocol (not a transmission protocol). As built, frame relay networks tended to ignore the “multi-point”, so frame relay became at once an *access circuit implementation* and *a network defined by using switches/routers that were marketed as Frame Relay*. Every providers frame relay product was architected as (A) point-to-point access circuits of customer to network cloud, (B) a way of passing packets across the cloud, and (C) another endpoint access circuit. This came to be called a “virtual circuit”. In the original MCI Frame Relay implementation, the cloud was actually one of the earliest and largest OSPF-routed commercial IP networks.

So in our billing data model, there never was an actual circuit between the two customer endpoints. So there is no way to validate, against the network, the endpoints of the data. In reality, the association of the endpoints was at the **abstract product level** – not in the network. Certainly packets entered at one customer endpoint and exited at another, but they never followed any contiguous circuit path. So if we measured a packet going in at the origin end, we had no way of associating that with any specific packet going out the destination end [except externally by an identity label in the packet.] Still, the virtual circuit idea made a kind of sense to customers; enough so we, as telecom service providers, could bill them and receive payment.

> Today, this packet-in/packet-out problem manifests in a new and twisted way. How can you tell that the packet coming out one side of a network is the same packet as what went in the origin side of a network? Leaving aside quantum mechanics and information theory which we will ignore, this is one crux of the NETWORK DATA SECURITY issue.

This example explains how *expectations that are applied to networks and OSS distort the ability of current telecom professionals to really see the network*. If they are not actually seeing the network for what it is, how can they expect to make it do what they want? If you are putting energy, time, and resources into solving a problem which is not congruent with the actual network, of course the results will not be what are expected and everyone will be quite frustrated.

This external circuit model was applied to Frame Relay performance reports and billing. The counts and classifications of packets were made internal to the switches; these were not consistently defined, or more exactly, were defined for traffic management reasons and not based on service provider product-driven definitions. Traffic counts for performance, pulled from traffic management parts of the MIB (Management Information Base) were done via different internal mechanisms than the packet counts for billing. I cannot begin to tell you how much time and resources were wasted explaining why these Frame Relay bills and Frame Relay performance reports never match up. This made customers very unhappy.

This external circuit model was applied to inventory. The concept of a customer's *virtual network* was completely the artifact of meta-data in product inventories. There were no distinguishing characteristics in a network that ever separated customer traffic, one from another.

This external circuit model was even applied by some to traffic management models and network design. This was particularly troubling when it came to calculating circuit utilization and applying this to determining how much space was left in a customers virtual network. TDM circuits had a straightforward way of considering utilization, how many of the intervals in which data could be placed on the wire, was data placed on the wire. The intervals were of constant rate amount so the more that were filled the better the economic performance of the circuit.

But virtual circuits (imaginary to start with) did not perform this way. They were deeply dependent on queue buffers. I will not go in to the why here, but it turns out that virtual circuits perform best when there is a backlog of packets waiting in the queue and when the total size of the packets shipped in a time interval is significantly below the computation of circuit bandwidth times time (see good-put and traffic management). Yet we reported Frame Relay virtual circuit utilization just like TDM utilization. Then the customers would get upset that they were not getting their money's worth when virtual circuits were running at only 60-70%, when in fact, this was the optimum utilization rate to get the most packets from one side of the network to the other.

Packets are dropped in *queue services*; the protocol is designed for this. As part of traffic management, packets are assigned a priority, based on QoS/CIR and then dropped preferentially by that priority. But "dropping things" has a *bad normative language value*, so we biased the customer perceptions against the service, just by how a normal technical operation was casually named. As a result, among customers and even telcom executives, there was a surprisingly large degree of suspicion that the networks were badly designed; because "any packets dropped" was bad engineering or operations. When marketing executives sought someone to punish for the dropped packets, battles erupted between Operations and Engineering because each truthfully claimed "I've done nothing wrong," and erroneously concluded "it must be him". Indeed, some customers came to believe that telecom service providers were dishonest and bought external probe products to check the networks themselves – but these probes used the same fallacious model and inappropriate language, validating what was actually a falsehood. **The cumulative loss of trust in service providers and networks was a deeply troubling trend with extraordinarily large, later negative externalities**.

Eventually, Frame Relay performance reports, filled with packet counts and utilization % for virtual circuits, got to be so large they could not even be emailed to customers. When customers got them they needed days to review them. So they bought external software from 3rd party vendors, just to rank and order the reports to tell them which circuits were in trouble. Even through data was passing fine from one place to another. I am personally angry at how much money was spent unproductively on performance reports; and at how dishonest the performance reporting vendors were in *selling to customer fears* based on problem utilization rates and dropped packets. And throughout all this, the network worked very well, as long as you did not poll for billing or performance information too often.

Remember Olympic traffic classes (Bronze, Silver, and Gold). There were not two independent parameters which could be set in the Frame Relay switch that could generate 3 distinct traffic classes in Frame Relay switches. In fact, in some cases, these were just separate network installations with better or worse network performance. In one case, mergers dictated that a network with lower QoS was labeled as a higher Olympic grade service. Another case, the lower Olympic class network, with cheaper prices to customers, cost more to operate than the higher class network by the same provider. It cost more to send a packet for cheaper returns than to send a packet on the higher class network. It was difficult for telecom companies to discover this about themselves, because the visualization of the product model had business precedence above the network dynamics; the product definition obscured the actual economics and performance of the networks. Because the product model said Gold is better than Bronze, it must be so…

PS: The proper billing model for Fame Relay (IMHO) would have been to charge based on access circuit capacity (specifically TDM capacity when a TDM access circuit into the cloud). Nothing more. Everything else should have been a tunable parameter used to get the best good-put (packet count delivered to the other side) possible. Specifically, no billing or product mention of CIR; always set CIR at the optimal point of the good-put packet transfer calculation. The customer should never have known about engineering things.

Part 3: Modeling the wrong map

"We read unconsciously into the world the structure of the language we use."
- Alfred Korzybski

Appling "Map is not the Territory" to data modeling:

The prior Frame Relay Billing example (This blog: *The New Telecom Ecosystem, Part 6* & *Complex Systems & Autonomic Networks, Part 2*) also underscores significant issues with standard, universal data models. These models seek to abstract similar features and build root organizing principles from which other more detailed objects and features will be derived. This abstraction to principle or root features is fundamental to achieving the unification promised by a data model. But the source idea from which abstraction is pulled is past networks and experience – picking from the past data and experience cannot reliably predict a future network organizing principle.

Models which rely on circuits as high level organizing principles can never represent packet routed networks. John Strassner in originally designing DEN recognized this, but few users of DEN do. But there are more subtle implications. (Picking again on the older stuff) An issue with TINA modeling was its reliance on past architectures as a unifying principle. Now TINA worked very hard to abstract out biases from the past and the then current ways of representing networks as layers and circuits. So they abstracted the notion of a circuit into a path, and represented paths as ordered lists of other path links (almost but not quite reaching the dynamite principle of recursion - too bad). This was a much better way of capturing what went on in Frame Relay networks than the virtual circuit model. But it still forced on networks the notion that networks built a road which channeled communication from one part to another – a product-oriented business goal, not an intrinsic part of the network. Paths were "nailed up" in the model; where in routed networks, each switch just knows who it is linked to and what links to put and outgoing packet on. TINA worked for ATM but not for IP, at least prior to MPLS.

In abstracting the traditional concept of layers in a network, TINA-C went too far. They tried to capture both the structural organization of a network and the layering of protocols in one concept. As such, TINA-C layers were always uncertain/fuzzy and often applied differently by different modelers. Linking one layer to another was the concept of a lower level path being associated with a higher level path. Again, if you step back, this is not a physical construct of modern networks. *Layering is not real*; what occurs is actually tunneling: Fiber carries optical information (SONET), SONET carries ATM which carries IP, or better, SONET caries IP directly. Ethernet links are different but can be similarly described. The only real connection is when switches of one network type are directly connected to another via protocol matching boards connected between switches/routers.

There is no real identity of a SONET ring with an ATM virtual circuit with an IP/MPLS path. In fact, to make networks resilient, you must discard the notion of a solid association; this should be a decision of the network as each technology seeks to maintain its own continuity during outages. In autonomic restoration time (microseconds to seconds), you cannot have a central network manager trying to tell the ATM network to switch in a transactional coordination with the SONET cutover.

What might work is a distributed systems model (not data/information model) where the action of restoration was embodied in a software agent who maps the same rules as the SONET switch; and another agent which maps the ATM restoration rules. Each follows the switching actions and time of the other; but in this case there might be time for the SONET agent, monitoring a switchover of its switch, to send a message to the ATM agent that corrective restoration is accomplished before the ATM switches. Then the ATM agent could intervene to keep the ATM circuit from jumping unnecessarily. Or else could coordinate an automated switch from where the ATM virtual circuit landed to the restored SONET path.

Of course, real, modern networks are more complex than this. Each network has its own technology and rules for QoS. Each might make separate decisions on the acceptability of a route. Perhaps it is better to let the decisions remain independent and float the network. Again, an agent can be complex enough, and identified in its rules closely enough, to adapt to these policy details. Distributed agents model the system accurately, precisely because (1) the agent talks directly to the device and (2) agent talks directly to another device agent if and only if the network switch is directly connected to another network switch.

> Any organism must be treated as-a-whole; in other words, that an organism is not an algebraic sum, a linear function of its elements, but always more than that. It is seemingly little realized, at present, that this simple and innocent-looking statement involves a full structural revision of our language...
> - Alfred Korzybski

Distributed Agents were a key component in the origin of the Fine Grained approach to network management. Each switch has its own agent and agents interact with other agents and with higher management services. Instead of the current trend of massive element managers doing everything possible for a network domain and costing millions of $; you have essentially a cheap *autonomic element manager for every switch*. A group of agents/services will automatically find each other and dynamically associate into interaction communities - based on the switching/routing parameters in the interlink boards of the monitored/controlled device the agent is modeling. It does not matter to the model or the agent, if the parameters are pushed down into the device (provisioning/configuration) or pulled from the device (performance, state management, or inventory) – it is all the same. So old FCAPS silo organizing functions are irrelevant.

In the Fine Grain approach, the intelligence lies in the self-organizing system of agents/services, not in the data model. No overall data model is needed. Each agent/service only needs to know how to communicate with the immediate community with which it associates. Because decisions are distributed, the information does not need to be funneled up to one common description for action by a centralized management application. The agents are distributed, distributing the policy information, instead of the data being shared out via a common central data repository.

In the Fine Grain approach, interfaces become more important than information models. Because communities can be localized and specialized, an agent in another community need not understand anything in common with an agent elsewhere. Hypothetical example, the SONET switch agent for Verizon can be a completely different implementation from the multi-service switch agent for Massergy. They only need a translation agent where the networks interconnect to perform secure distributed decision making about coordinated network actions where they intersect.

Macro size SOAs (Service Oriented Architecture) can also use this principle. (IBM is a leader in applying and designing macro agent-based SOA's; however, they discontinued efforts at applying these to networks.) The interface is still the key and the domain of common knowledge and similarity need not extend beyond immediate interface interactions. But in these cases, unified data models are often needed to coordinate macro-scale business principles. But here the 80-20 rule works: unifying 20% of the information satisfies 80% of the business needs.

Now there are still economies of scale that can be realized by making agents similar to each other. *Agents can be built from a similar template*. The agent can be specialized by (1) assembling in the needed interface definitions, (2) the local data interaction model and (3) by specialized policy. The interface establishes the dialects the agent understands. The policy determines the behavior of the agent. We will cover policy in later parts of the blog series.

There are also economies of scale in having a common information model shared by all (even Fine Grain). What I am emphasizing is that this common model, while valuable, cannot itself become a goal of architecture (as we once made it.) *It is only a means that must be selectively applied and held to rigorous scrutiny*.   While we scrutinize data models today, it is for correctness to the principles of data modeling. It is not always a *hard look at if the model is congruent with the reality of the modern network*, as it should be.

It is also NOT possible to future proof an information model. It cannot be detailed enough to be useful and still accurately predict the way future networks will be organized. A model may last a whole technology generation – but new models must arise for new network generations. And these could be fundamentally different. By fundamentally different, I emphasize that they may not be data models at all, but complex systems models.

Part 4: Catastrophic Problems

> "There are only two types of networks, those which have failed and those which have yet to fail."
> - Anonymous bar joke among network and OSS designers

To the best of my recollection in the passage of time and with acknowledgement that I did not have an active role in this example so some details may be wrong, this is another historical example.

Problem: Once upon a time, long long ago in the nineties, a major service provider's frame relay network went unstable. Customer end points started becoming unreachable. Each time, repair operators followed a NOC procedure list of probable cause and do-this-lists of tests, actions and scripts, all without result. While well down on the list, rebooting the router resulted in service restoration for the complaining customer. More and more of these isolations started occurring. Early on, it was difficult to see any causal linkage or pattern because new isolations occurred in distant parts of the network from where the original isolation occurred. Finally, the problems started moving in waves across the network with customer's endpoints becoming isolated and then spontaneously fixing themselves. More and more customers were affected until a big part of the customer base experienced this issue. The trade journal press got hold of the problem and lots of bad publicity resulted, eventually making headlines in the major press.

Response: Let's break down the services provider's response to this outbreak. A few weeks in, as soon as it was realized that a significant trans-network problem was occurring, Tiger Teams were established to find the problem. [A Tiger Team is a collection of experts usually spanning several organizations like Operations, Engineering, and IT, which are removed from ordinary job activities to work closely together to solve this problem.] Original network activity logs were reviewed and the team identified that a new router/switch load had been installed about a week before the problem started. The Lab's network tests of the current 'problem' load were reviewed and then repeated but resulted in "no problem found." Nevertheless, no other course of action being revealed, reversion to the old network software load was recommended. A trial program was started to return a subset of the routers in one zone to the prior network load. The problem did not stop.

This was very difficult for management to accept and the Tiger Team was reorganized. The enhanced team recognized the problem as "routing storms" floating through the network. Eventually the problem did not disappear until all of the several hundred switches were returned to the prior load. Restoration eventually fixed the problem, but several weeks of outage problems angered customers, and the press bashing lasted much longer than the actual network problems.

Vendor/Provider Interactions: During this entire time, as in standard practice, the problem had been escalated to the switch/router vendor to isolate the problem. Their lab tests were also inconclusive, so eventually a line by line review of all software changes in the routing code was called for. Finally it was found that a line of the OSPF routing code had been replaced with a debug line that was mistakenly never removed and carried into the network release. Major finger pointing started occurring between the vendor and the provider and long established trusts went down the tube.

Post Mortem: What had happened was this. To test routing update propagations, the vendor's switch programmers had changed the timing of OSPF updates from a longish period to a rather short period – this so they could observe the routing changes propagating and stabilizing. As everything was working in the test network of a dozen or so switches; a vendor supervisor OK'ed distribution of the update code (without the debug line being removed). When it got to the service provider network and was placed on their test bed of a dozen or so routers, all the script tests passed fine. The software was scheduled for loading in the network, a few switches every night. This was a good, cautionary approach to testing, qualifying, and deployment of the new code by the service provider. Eventually the network rollout was complete.

Problems in the real network began to be observed about a week later. But in retrospect, a post mortem of network and OSS logs found that problems began occurring when something-like half the switches were converted. But the seemingly random, unrelated nature of the problems went 'under the radar' as they were *each fixed in isolation* by the NOC "reboot the router card" procedure. These tables are switch control systems for which of the many outward bound links to specifically palace each packet. In routing code whenever a change or reboot occurs, routing update messages are sent out, which cause each switch to respond with routing table "access to" entries, and the return messages are then received back at the rebooted router. Then the router computes its new routing table. Only the timing, and thereby the frequency, of update messages was changed. So what happened?

The problems never occurred in the test bed lab networks because they were too small and switches were too close together. An update message was sent whenever a change occurred in the router. Each update message caused a response from connected switches and resulted in the message sender recalculating its routing tables. With the smaller network, the number of responses was small so the time to re-compute the table was small. Network behavior was stable for smaller networks. *This problem could not be observed as behavior in a smaller network*. Even debugging code inserted to specifically watch for this only discovers if the frequency and time of updates is inline.

My Post Mortem: The problem manifested when two things occurred: (1) the networks became large enough, in number of connected routers, that the computation times were very long; and (2) a return request for a routing update from another switch occurred before the re-computation from the earlier request was complete. This restarted the computation problem and sometimes, but not always, isolated the links which had not been updated. The problem became major with NOC responses. It turned out, that *the NOC action of fixing the isolated links, for which no apparent problem could be found, by rebooting the router was a major participant in the cause of the network failure*. Each reboot caused a new update to be sent out, isolating more links. A positive feedback loop started.

Every attempt to fix the problem caused more problems to occur, at random, and distant from the fix. Eventually, the number of request messages and re-computation starts was so large, that a new stability point was engendered in the networks and the problem was self sustaining. Customers came in and out of service as the routing storm raged across the network. This network was sufficiently large in scope and extent that the storm was self sustaining. [A new Markov Mixing stability point had occurred - more on this later.] This is why fixing one section of the network with the old ‘good’ software load did not work, because the old load was subjected to these same forces.

The original, simple change in the frequency of update requests in debug code had potentially destabilized a network. This change in update timings caused occasional link drops in distant switches inside a large network with significant geographical distance between switches. Left alone, a few problems continuously would have occurred, and then later fixed themselves; but not a large number, because routing code is written to suppress this kind of behavior. But the NOC procedures for response had feed the problem by causing many reboots of the switches thereby sending the number of routing update requests above a threshold, causing the catastrophic response.

At the time of the problem and solution, even getting the NOC to acknowledge that they participated in the problem was difficult. The assessment presented here was not accepted officially. Finger pointing continued between vendor, engineering and operations.

Costs: This was one example, but in the nineties, every major Frame Relay service provider experienced some form of catastrophic network systems behavior. The costs of these failures were enormous. Outages occurred for days and good will was lost. Customers switched networks and sometimes entire accounts, with all product groups, were affected. If you include the costs of the failures to the customers, whose internal operations were affected, the costs increase by orders of magnitude. If you include these external costs with the costs of operating a network, than a truer cost of OSS design and network design can be reached. But these catastrophic and customer externalized costs are never quantified into the “efficiency computation factors” of design and purchase decisions. “Lean and mean” decisions will result in suboptimal cost allocations and losses in the network.

Complex Systems: This Frame Relay routing update failure example is presented as a lead in to the next section. Networks are now so complex and so large that they exhibit specific system level effects. These effects are not evident in the micro behavior of the individual network components. Also, we must be broad in our inclusion of “what is the network”. Actions of the NOC and OSS with their procedures designed around **failure incidents**, helped cause the problem. So NOC and OSS both must be included as **actors** in the **complex system** if we are to understand network behavior.

It is hoped that these blog articles will began to change this. In the next section, we will look formal study of complex systems and emergent behaviors.

Part 5: Emergence in Networks

> "…, networks evolve over time through the actions of network users, changes in types of access to the networks, and interconnections between networks. … changes to one part of the network can affect the performance and evolution of the network in ways that can be dramatic and unpredictable."
>
> - Daniel F. Spulber & Christopher S. Yoo

Instability: It is actually difficult to understand, to realize, when a network goes unstable. In the beginning a larger than normal series of problems will occur. These problems are treated in isolation and standard fixes are applied. These fixes either do not take, or the problem just shifts to somewhere else, often an adjacent area. The number of problems rapidly increases, showing similar symptoms, but no rational cause for why this is occurring.

A NOC just following procedure scripts will only see the individual problems. Centralized systems must add up the total number of problems per unit of time, class them as similar, and then trend them to give early prediction of the onset of an unstable network occurrence. Regardless, eventually this becomes obvious, when the point is reached where the network cannot do its job or even shuts down. How can we best see these problems as they occur and better yet, build systems which accommodate and correct for these problems?

> "**Emergence** is the process of complex pattern formation from simpler rules… An emergent behaviour or emergent property can appear when a number of simple entities (agents) operate in an environment, forming more complex behaviours as a collective…. Thus it is not just the sheer number of connections between components which encourages emergence; it is also how these connections are organised. A hierarchical organisation is one example that can generate emergent behaviour (a bureaucracy may behave in a way quite different to that of the individual humans in that bureaucracy); but perhaps more interestingly, emergent behaviour can also arise from more decentralized organisational structures, such as a marketplace."
>
> http://en.wikipedia.org/wiki/Emergence

Studying complex systems and emergent behavior will give two specific powers: (1) a better ability to visualize what is going on in the network as a whole and within any network subpart; and (2) an ability to design stable complex systems. In the following, we look at the structure of emergent systems, designing for stability, and introduce **agents** as design patterns for network OSS.

> "Emergence is what happens when the whole is smarter than the sum of its parts. It's what happens when you have a system of relatively simple-minded component parts -- often there are thousands or millions of them -- and they interact in relatively simple ways. And yet somehow out of all this interaction some higher level structure or intelligence appears, usually without any master planner calling the shots. These kinds of systems tend to evolve from the ground up."
> --- O'Reilly Network Interview with Steven Johnson

Visualization: Most of the time, emergence and complex systems are approached as mechanisms for supporting very large systems. Complex systems are designed to accommodate very large structures.

> "Emergent systems… would not be overwhelmed by this volume and are, in fact, a natural solution to such hugely complex problems. The question then becomes, do our current systems, business plans, and curricula reflect our understanding that hierarchical systems are only a temporary solution, and are we ready to implement the systems, business plans, and curricula that will be a part of the emergent technologies that will be useful and necessary in the near future?"
> --- from a review of Johnson's Emergence by Chris Leslie

Yet there is not just a scaling problem for OSS products and systems – there is a visualization problem.

Much of the problem with understanding and fixing networks is the sheer size of modern networks. Much effort is spent just documenting additions and deletions and drawing network maps. Sometimes as we have shown, the maps themselves distort perception of the network, but even with perfect maps, there is just too much there for people to comprehend. Sometimes the best experts are said to "have a feel" for the network. Generally these operators, without being able to explain how they know what to do, can put their finger on a problem and suggest a approach to fixing it. I maintain that these operators have subconsciously grasped the complex system dynamics and are responding to that, rather than to details. This is why it is so difficult for knowledge experts to generalize top-down rules from speaking with these rare operational experts.

Stability: The most interesting aspect of *Emergence* (to OSS & network builders) is that emergence often leads to stability in complex systems. The most interesting property is goal directed behavior can lead to emergent stability. Emergent systems design builds actor agents which push back against instabilities - so that any displacement from equilibrium or noise introduced into a system destabilizes the system only for a short time, before stability is again achieved - without external action.

> "A hierarchical, top-down system attempts to use a centralized decision-making process based on abstract rules to guide behavior. The emergent position looks at complex systems differently: a small number of rules that are processed by individual units are the best method of explaining the aggregate behavior. While a statistical analysis of an emergent system will lead to abstract mathematical laws, these laws do not explain why individual units behave the way they do."
> --- from a review of Johnson's Emergence by Chris Leslie

Establishing system boundaries: When designing for autonomic networks, agents introduced into the system would act to restore the system dynamics. Some think that in a fully automated network it would not be necessary to call upon operators in a NOC to take action to restore the system. The actions of the software agents would automatically bring back stability. But getting the complete set of rules for these agents is extremely hard and the network and product mix is changing all the time.

Alternatively, one can consider the operators in the NOC to be part of the autonomic network system. In this view, operators are 'living agents'. As such, ideally, their actions should be able to be encoded as rules; but we know this is extraordinarily difficult. This is partly why rule based systems when applied to network alarm identification and response; do not give the good results that were expected.

Instead, following our understanding of complex systems, we could use a number of different disciplines to model the actions of the NOC operator – depending on if we wanted to be proscriptive (please behave this way) or prescriptive (tell me how you would behave in this situation to get this result). Basically, in *classical OSS*, designers build a script for operators to follow for customer relations or a process to follow for network restoration. But in reality the operator will not always follow these rules and process, often diverting because new situations arise, or even because they are bored and begin to experiment (a normal human behavior). In this case fuzzy logic is needed to model the operators.

Fuzzy logic: Basically fuzzy logic lets things fall into multiple sets or partially reside in a specific set. Programmatically, it involves using rules of “If Then” form and not rules of “If Then Else” form; and many rules can be true at once. In building an OSS agent, we can attempt to model and program all the behaviors a NOC operator might take or has taken in the past. But how would a specific action be selected from the many possible actions during run time? For this we might apply Bayesian utility theory and assign weights to the actions. These weights help conscribe the action of the model operator agent via internal agent control rules: such as by weighting the probability function when random selection is used to chose which of the many true rules to implement this time. Such designs allow evolving systems to occur.

Iterative design: In evolving systems, the weights for these rules will change over time based on the perceived, in aggregate, degree of successful actions that the rule participated in. Of course, the trick is recognizing success and then programming this feedback of changed weights. Recognizing “what is success” is itself a fuzzy construct with many values to the set. In this design model, by small iterative improvements, over time the network system will become stable; the rule weights reasonably mature and changing only in small amounts. With small changes to agents whose action is localized to a specific subpart, interactive changes cascade through the system. If the criteria for weighting the rules were empirically measured and success- based, emergent stability is achieved. But predicting outcomes of rule changes, prior to implementation, is difficult.

Alternately, we can cluster associated rules into generalized behaviors. We then put each separate rule cluster (a subset of all rules describing possible behaviors) into a separate agent. By describing some form of inter-agent, top-down control (such as one of several types of transactions, or sequence calls, etc.), we attempt to gain a systems dynamic in which emergent system behavior arises.

> “There is nothing that commands the system to form a pattern, but instead the interactions of each part to its immediate surroundings causes a complex process which leads to order.”
> - [http://en.wikipedia.org/wiki/Emergence]

Self organizing & self provisioning: The best agent managed system will be self organizing. If the agents can themselves determine what other agents to interact with in every instance, the overall network system is beginning to become truly autonomic. There is reason to believe this would deliver an extremely stable, inexpensive to operate network.

> “In some cases, the system has to reach a combined threshold of diversity, organisation, and connectivity before emergent behaviour appears.”
> - [http://en.wikipedia.org/wiki/Emergence]

Having only a few agents will not result in emergent stable networks. Classical OSS, combined with Enterprise Resource Planning (ERP), and the postulates of the “lean operator” have resulted in *reducing* the number of management systems and NOCs in most providers. It is paradoxical, but while this will reduce immediate operational expense, ultimately, consolidation of NOCs and management systems will increase costs. Over the long run, NOC and ERP- based application aggregation lead to a greater composite sum of costs than having a large number of simpler, specializing NOCs. This is counterintuitive; but consider the simple case of the rare, extreme disaster such as a routing instability or security breech. The costs of recovery can run large enough to add significantly to the life-time sum of costs of the network. The probability of a destabilizing decision is greater when one entity makes the call, rather than a number of entities debating the call. Remember the old adage: "There are only networks which have catastrophically failed, and networks which have yet to catastrophically fail."

But many NOCs, many applications, many agents, many alternative actions an agent can take - this is complexity. If you attempted to manage this with traditional methods, the sheer complexity would overwhelm you. There must be a class of autonomic management agents that oversee the provisioning and health of these agents. This then leads to **survivable systems** design. One ends up with a "Mirror World" composed of a complex system of agents that models a complex network.

This was a fast overview of emergence and agents as actors in emergence-aware design. In future articles I will outline a revisualization of the network so that emergent behavior can be identified, while showing the fallacy of the point fix; and how to build autonomous agent systems, as described above, that could enforce stabilizing behavior in networks.

See the 2007 articles on "Autonomic Communications" and "Self- Systems"*

- *End* -