A Knowledge Plane for the Internet

David D. Clark, Craig Partridge, and J. Christopher Ramming

v4.6 of 2/5/03

Abstract

One of the Internet's greatest strengths is that it does not know or care what its applications are or what they are doing: it simply forwards data. Yet network users experience the network through the functioning and performance of applications. This divergence of perspective leads to a number of problems. For example, a user whose local DNS service has failed may perceive the network as broken, even though from a network perspective, data continues to flow correctly. If an email server or a Web server fails, the user will say the network is broken; the network operator will say the network is fine.

We need a way to make the network more aware of itself and its applications, without destroying the open and transparent data plane. To meet this need we propose the creation of an Internet *knowledge plane*. The knowledge plane is a distributed and decentralized construct within the network that gathers, aggregates, and manages information about network behavior and operation, and provides an integrated view to all parties (operators, users, and the network itself). The goal is to enlarge our view of what constitutes the network to match the intuition of a user, and to enhance our ability to manage the network intelligently, without disturbing the open and unknowing forwarding plane.

The knowledge plane is intelligent: it can reason about the network's behavior and act upon the results of its reasoning. It can remember and learn from past behavior. To achieve that goal, we propose to adapt and employ recent work in cognition such as the separation of algorithm, policy and goals, and new models for knowledge representation.

Introduction

The Internet's success is also its weakness.

The Internet's data plane is simple, transparent, and open to new applications. This design is a success—because the Internet does not constrain the applications that run over it, the Internet has been a platform for innovation, and the creation of new value. The core of the network has, by design, no idea what the applications above it are doing, or what the traffic patterns on the net signify [1,2]. That simplicity and transparency is why the network can be open to the easy deployment of new applications.

But exactly because the core does not "know what is going on", it cannot easily detect when something has gone wrong. Normal email, spam and email with a virus all look like packets to the data plane. If the Domain Name System (DNS) is unavailable, a user of the Internet will say that the Internet is down. Similarly if a web server or email server fails, the user may conclude the network is down. But the data plane may still be working just fine, and an Internet operator will conclude that the network itself is up.

So there is a serious disconnection between what the application's user sees, and what the unknowing core of the network sees. Viewing the problem abstractly, the network (and its operators) cannot detect deviations from acceptable behavior, because by design the network has no model of what is acceptable, and indeed, no mechanism for measuring behavior and examining it for acceptability.

At the opposite design extreme, the telephone system is a successful network that knows very well what is going on. It is marked by a complex and powerful core, very simple edge-devices (the phones themselves), and a clear model, built into the core of the network, of the application the network serves. The telephone system is designed knowing that the purpose of the network is to carry telephone calls. Since the network has a model of what its purpose is, it can do diagnosis, fault isolation and, in some cases, automated repair of the total system. But this ability to "understand" the total service that the phone system delivers comes at a cost: adding new applications to the network is difficult unless they look like just another telephone call. It is very hard, in any interaction with the telephone system, to avoid working in time units that are multiples (or fractions) of 125 μs or communications abstractions that are not circuits.

Can we capture the best of both designs? In our view the answer is clearly "yes" but the solution is not simply an averaging of the two techniques to create a network of middling intelligence. A new design for the network is needed that marries the transparency and utility of the Internet with a new approach that captures an understanding of what the network is doing and what it should be doing. This approach should understand application-level behavior, high-level representation of configuration, management, and control.

A proposal: the Knowledge Plane

The simple and transparent data transport plane of the Internet has been a success, so this idea should not be discarded. Instead of trying to compromise and add some measure of application knowledge to the data transport plane, a better alternative is the construction of a separate **knowledge plane (KP)** in the future network. The knowledge plane is a distributed and decentralized construct within the network that gathers and aggregates information about network operation, and provides an integrated view and a consistent set of control signals. It can embody information from all the traditional layers of networking, from the application down to the physical layer technology. It sits at a higher level of abstraction than existing management tools (hence the proposed name): it embodies high-level models of desired behavior, global sharing of information, and rules that map observations and assertions about network conditions to explanations and responses. It has the ability to learn from prior events, so that its ability to explain and control improve over time.

From one perspective, the knowledge plane already exists in the network today, in the heads of network operators and users, who form a global network hooked together by email and phone, as well as in management data collected by network elements. Some knowledge may be online, but is considered proprietary by network operators. From this perspective, the knowledge plane has the goal of automating more of network deployment and operation, and reducing the labor-intensive manual procedures of today.

From another perspective, the knowledge plane is an intellectual innovation. Traditional network management systems are "bottom-up" and regionalized—each system manages its assigned part of the Internet. In contrast, much of the "knowledge" in the knowledge plane is produced, managed, and consumed near the edge of the network. The participation of applications (as well as other protocol layers such as transport and physical) is what would enable us to achieve an application-specific understanding comparable to that of the telephone network, in a network with an ever-increasing diversity of applications.

What is the Knowledge Plane good for? Two examples:

Example one: When some part of the Internet fails, it is almost impossible for the end user to tell what has happened, to figure out who should be notified, or what to do to correct the fault. The distributed operation of the Internet means that the network operators may not be much better off. Imagine a program (we call it the *why* program) that a user can run when something about the network or a networked application seems to be broken. So, for instance, the user might ask, "Why can't I get to www.acm.org?" The *why* program starts with a component that runs on an end node, and performs diagnosis when there is a failure. The diagnostics can check out functions at all levels, from packet forwarding to application function. There are several current research projects that this application could build on [13,14].

Once the end node has performed what diagnosis it can, the next stage is for the tool to add assertions to the shared knowledge plane about what it has discovered, and ask the knowledge plane for relevant information. This contribution to the knowledge plane allows all the users on the network collectively to build a global view of network and service status. This data can be combined with information derived from measurement efforts now going on across the Internet that attempt to build an overall model of network status [9,15]. Using this information, this tool would give the user an explanation of what had gone wrong in terms that are meaningful to the user, and also information to the network operator in his terms. Network operators have the option of adding additional facts to the knowledge plane about known failures; in the ideal, a user who trips over a problem might not just get back diagnostic information, but information from the provider about when the problem will be resolved.

Example two: The dynamic routing of the original Internet did not take into account administrative and policy constraints, so routing today is more and more defined by static policy tables. This means that devices such as routers are increasingly manually configured and managed. Static tables and manual configuration make the network brittle to failure, hard to change, and even harder to reason about globally. Imagine a distributed configuration manager for a region of the Internet, which would accept high-level assertions, at the administrative level, about how the components of a network are supposed to arrange themselves, and guide the actual detailed configuration accordingly. Examples include controlling the deployment of a consumer network in

the home, an ad hoc network in support of a rapid deployment force, or a network for a small business.

The distributed manager should have enough understanding of low-level configuration to detect if the network is properly configured according to the high-level constraints, to detect if a better configuration alternative is available, and to detect if the system appears to be corrupted. The system must be able to deal with different assertions made by different parties, and either compose them or detect that they are inconsistent. Successful accomplishment of this project could lead to substantial reductions in manpower needed to configure and operate networks.

Previous attempts to do "high-level network management" have not been very successful; one possible reason is that previous projects have not been able to find the correct high-level abstractions. The necessary hypothesis is that there exist suitable ways to abstract detailed behavior, and to talk about goals, plans, constraints and methods at a high level. The knowledge plane is much more that a data-base of facts—it is a construct that embodies cognitive tools and learning.

A Knowledge Plane Architecture

While any proposed knowledge plane architecture will inevitably be changed and improved based on experimentation, it is useful to briefly explore what architectures the preceding discussion suggests.

At its heart, the knowledge plane will require the creation of a distributed system in which the knowledge plane will reside. Components of this system will run on edge computers and on servers placed around the network. The system must support the creation, storage, propagation and discovery of a variety of knowledge plane data, including: *observations*, which describe current conditions; *assertions*, which capture high-level goals, intentions and constraints on network operations; and *explanations*, which are an example of how knowledge itself is embodied—explanations take observations and assertions and map them to conclusions.

In addition, the knowledge plane needs to provide a way to think about, access, and manage what the cognitive community calls *sensors* and *actuators*. Sensors are entities that produce observations. Actuators are entities that change behavior (e.g., change routing tables or bring links up or down). So, for instance, a knowledge application that sought to operate a network according to certain policies, would use sensors to collect observations on the network, use assertions to determine if the network's behavior complies with policy, and, if necessary, use actuators to change the network's behavior.

Above the basic building blocks of the knowledge plane are a set of KP services, and above that are applications that run on the knowledge plane: K-apps.

Knowledge Plane Services

Part of the KP will be a set of services that are provided to support knowledge-based applications. We discuss two examples here.

The first example is a service that captures the trust relationships among the parties—who has decided to trust whom. We illustrate this "web of trust" feature using an example application—control of spam. As an experiment in adding application information to a knowledge plane, imagine an enhanced email architecture that can control the delivery of unwanted email (spam). The proposed approach is a combination of a number of techniques that are being tried today, together with the addition of a "web of trust" to the knowledge plane. Use some mechanism by which senders can sign mail, so that the "From:" field cannot be forged. Have each user build up a private data-base of users that he is prepared to talk to. If a user receives mail from an unknown sender, build a bottom-up peer-to-peer mesh of parties that agree to trust each other, into which a query can be launched to try to validate the identity (friend or spammer) of the unknown sender. If the sender is not known to any of the receiver's "circle of friends", treat it tentatively as spam. A second layer of shared knowledge can then be used to see if anyone has verified that this message is spam.

This service would require the development of trust models, and the use of scalable techniques (such as the so-called "small world" models [10]) to search a web of trust. A web of trust, once constructed, could be used to improve the coherence of other applications. For example, if a user is making a query of the knowledge plane about a possible network fault (as part of the *why* program, for example), he will be more inclined to believe an assertion from a node he trusts. If a previously trustworthy node is generating spurious information, this may be a signal that it has been corrupted. So trust can be both exploited and validated across applications.

The second service is a data aggregation service. There are many tools being used today that measure some aspect of the network—latency, loss rate, route instability, forwarding failures and so on, to try to determine if there are problems of various sorts. The knowledge plane will provide a framework and mechanisms to combine these observations into one or more global views. The knowledge plane will have enough information to detect inconsistent measurements (which might signal a broken or compromised network region.)

Knowledge Applications

The *why* program and the high-level configuration program are examples of K-apps—applications that run over the knowledge plane, making use of its facilities and the information and rules that it contains. Here are some other examples:

Support for overlay networks: We are increasingly seeing the development of application-specific overlay networks on the Internet. Each overlay network uses edge-based mechanisms to evaluate the performance of different possible paths through the Internet, and seeks to build a set of transport paths that effectively route application packets through what appears to be the part of the Internet best suited to the application's needs. Currently, application networks must probe the Internet, because there is no mechanism for them to learn about the capabilities of the network core. There is a risk that *ping* packets will come to dominate network traffic. The knowledge plane would be in a position to aggregate application- and network-derived knowledge about network performance, offer applications better information about the network than they could learn by probing, and access to control points whose behavior could be modified to help better meet the applications' needs. The knowledge plane thus enables per-application control over traffic without the need to build per-application infrastructure throughout the network.

Shadow routing in the knowledge plane: This project is an example of building a knowledge plane application with knowledge of what is *supposed* to be happening. Today, routing computes a "best" route to Internet destinations. But there is no computation of acceptable alternative routes. More interesting, there is no computation of the best route *from* any address. The design assumption of the Internet is that if a packet from a given source arrives over a given interface, then this must be a legitimate path from that source. It is this assumption that allows malformed routing assertions to deflect packets to distant parts of the network, allows DDoS attacks to contain arbitrary source addresses, and so on.

This knowledge plane application would complement the actual routing protocols of the Internet with a "shadow" computation that runs in the knowledge plane. Autonomous Regions (AS's) may make assertions about what connections are acceptable for traffic flowing to and from them. These assertions can be combined with the BGP assertions seen at various places in the network to build up a model of what acceptable routing options are. This can be used to detect potentially invalid BGP routing assertions and false source addresses.

Knowledge-based intrusion detection: There are a number of projects (and a number of products) that perform some sort of analysis to detect network intrusions. In general they look for patterns in data observed somewhere in the network. The current generation of these tools generate both false positives and false negatives. It has been hypothesized that a next generation of tools for intrusion detection will require that observations from several points in the network will have to be correlated, in order to get a more robust and useful signal. The development of the knowledge plane provides a basis to implement this data gathering and correlation.

While one could build the infrastructures for all of these applications separately, integrating them into a knowledge plane would provide considerable leverage. The user experiences (and their causes) recorded in the knowledge plane by the *why* application could be used by the configuration program to detect configuration problems. Similarly, an application overlay network could use both *why* and the configuration program to improve the design of the overlay network (using configuration policy to better place overlay links, and feedback from *why* about how well the overlay links are serving their application).

Challenges to Creating a Knowledge Plane

If we are to create a successful knowledge plane, we must grapple with and solve a number of challenging problems. Because one of the goals of the knowledge plane is to give applications the ability to learn and reason about their environment, many of these problems sit at the boundaries of networking and artificial intelligence.[1] This section sketches some of the key themes that run through these problems.

How do we represent and utilize knowledge? We want the knowledge plane to support reasoning (figure out why John can't reach www.example.edu) and learning (the last time www.example.edu was unreachable, there was a DNS problem, so let's check DNS performance). The current state of the art in reasoning and learning tells us that we need to build abstract models of the entities we seek to understand, and then use information to reason about, and potentially update, those models. Current research into schemes for representation, such as the DAML Project[2], may give us some insight into how to represent information

[1] For a general overview of knowledge representation issues, the reader may wish to read [16].

[2] The DARPA Agent Markup Language is a set of extensions to Extensible Markup Language (XML) and the Resource Description Framework to support ontologies (statements of relationships between objects) for web objects. See www.daml.org.

about which we can reason. However, we must also work out how to extract and process all the valuable information that presumably is not in DAML (or whatever form we pick) but in SNMP MIBs, system logs, and other disparate places. How do we construct, represent, and distribute the models that drive the reasoning?

How do we achieve scalable utility? The knowledge plane is a building block for a network that is more reliable, more robust, and more secure. Properly implemented, it should continue to improve the network, even as the network gets bigger and the knowledge plane itself gets bigger. As we add more knowledge and new applications to the knowledge plane, it should become more valuable and useful overall. Those are hard goals: as the volume of data increases, or the number of elements in a system grows, we all too commonly find bottlenecks and algorithms that do not scale. For example, if a network failure triggers a flood of messages into the KP, how are these aggregated and controlled so that parts of the KP are not driven into overload? We will likely find ourselves challenged to abstract data and impose compartments or hierarchy on portions of the knowledge plane to allow it to scale – how do we ensure that the abstraction and compartmentalization adds rather than subtracts value?

Finally, there is no way to expect that the information in the knowledge plane will be consistent or complete. A system of this scale must be predicated on the assumption of information that is partial, inconsistent, and out of date.

How do we route knowledge? Suppose the knowledge plane learns a valuable new fact, or comes to a valuable realization. How is that fact or realization disseminated? Is it pushed out to all interested parties? If so, how do we know who the interested parties are? Is the fact simply labeled and placed into the knowledge plane for interested parties to discover? If so, how do the interested parties know to look for it? Are there ways to intelligently summarize data that make these push-pull tradeoffs easier (e.g., so that an event such as the Baltimore tunnel fire produces one outage report for the entire network, rather than an outage report for each ISP, or worse, each link, that is present in the tunnel)?

How do we provide the right economic incentives? The networking community has come to learn that the success of a distributed system depends, in large part, on the economic incentives embedded in the system's design [11,12]. The knowledge plane is rife with economic challenges. How do we motivate people to put information into the knowledge plane? Much of the data in the knowledge plane will be valuable – should the knowledge plane provide mechanisms for people to buy and sell information (or better, "knowledge")? How do we avoid making the knowledge plane protocols a point of economic competition (e.g., avoid the vendor-specific enhancements to HTML problem)?

The knowledge plane represents a way to develop an overall view about what is actually happening, even if operators do not want to reveal their internal information. Some operators may not want to contribute fully to the knowledge plane, but if the plane is successful, it should generate more benefit than cost to each player.

How do we deal with malicious and untrustworthy components? There is no way that we can expect that all nodes in the KP are trustworthy, competent or reliable. Broken nodes may inject malformed observations, some nodes may lie about their behavior, and some players may attempt to disrupt or confuse the KP, either as a way to attack the network as a whole, or to gain some advantage over others. How can the algorithms of the KP protect themselves, filter out bad information, and reach valid conclusions in the presence of uncertainty and misrepresentation? The KP system will have to depend on approaches such as consensus, rating, and cross-checking to detect mal-formed or malicious behavior. A design that is robust to inconsistent inputs is necessary for success.

The security architecture of the knowledge plane must be designed in from the beginning. A poor security architecture will doom the idea; on the other hand this is an opportunity to try to build a new distributed system that takes security into account in a planned way.

As proposed above, a model of trust should be a core building block of the KP. Building a model of trust requires that there be some persistent robust expression of identity. There is no requirement that the identity be linked to a actual person (although for some purposes this may be preferred); the minimum requirement is that identity not be forged or stolen, so that one can build up a consistent model of trust based on prior observations of that identity.

A Detailed Look at Knowledge Routing

Physically, the knowledge plane is a distributed system that runs on machines around the network. To capture the instantiation of the knowledge plane, we use the term *think point* (TP) to describe the nodes in the knowledge plane. Considering the *why* program, it is clear that there must be a think point at every participating end node. The *why* program has to work when disconnected from the network, because one of the reasons that a user cannot reach a distant machine

is that the computer is not plugged into the network properly (or the network is down). So some small part of the KP must exist on any system that has a *why* program.

But in addition to the think points at end nodes, there will be servers scattered about the network, to which observations are routed so that knowledge can be brought to bear on them to generate explanations and responses. Routing of observations in the knowledge plane is a critical issue. There is no way an observation can trigger an explanation unless the two encounter each other in the knowledge plane.

The design of routing in the KP is a subject of future research, but here is a possible approach—one of several that may make sense. For a range of issues that have a *topological* nature, one way to route in the KP is by creating routes that mirror the AS mesh of the Internet. Thus, for example, if a host makes a *why* query because there seems to be a routing problem in the network, it might proceed as follows. Assuming that the end-point is properly connected to its local network, it runs a traceroute to get an approximate location of the error. Then it consults data maintained nearby in the KP to map this traceroute to a series of AS values. It then sends the *why* query to the TP representing the first AS along the path. If this *why* query is the first one of its sort that this TP has seen, it will forward it to the next, and so on. If the TP has seen similar reports, it need not forward the query, but can just report back to the sending host that the problem has also been observed by others and is being investigated. In this way, duplicate information is filtered out near the sources as it is injected into the KP.

Where would the TP be that is responsible for an AS? An obvious answer is that the administrator of the AS would run the TP for that AS. This may be a bad idea for several reasons. First, if the AS is down, this could render its TP unreachable. Second, the administrator of an AS may not want to disseminate the sort of reasoning and conclusions that might occur in this TP. A TP run by an administration about itself might have a one-way character—the facts go in but nothing comes out. This implies that there needs to be an external TP for each AS, which is run independently. It ought to be near the AS (why route queries about a US AS to Australia?) but far enough away to be autonomous. In fact, there could be more than one external TP for an AS, which redundantly reason about that AS and compare answers, to detect that there is a corrupted or disabled TP out there.

Is there a role for an internal TP for an AS? Of necessity, yes. First, each node in the network needs to be able to reason about itself. Second, a network operator will want to have a TP that represents its own interests, and there is no reason to prevent this. So for an AS, there will be one or more internal TPs, a small set of external TPs, and all these should receive reports and queries from the outside. This in turn raises the question of which of them report back on the reasoning they do. Just as there is aggregation of queries and observations, there may need to be aggregation of reasoning and conclusions in the other direction.

Another radical possibility is that there are companies that compete to provide the TP for a given AS. Each company collects its own data and sells its observations through a number of TPs. The *why* application would seek information from whichever company or companies it believes provides the most accurate and timely (or most cost effective) information. A knowledge marketplace creates a whole host of architectural challenges, ranging from how to reason about information from multiple TPs (even if three different companies tell you the same thing about an AS, it may turn out that they're all reselling data from one Internet weather service – if you really want a second opinion, how do you find the second weather service?) to how to design KP protocols to discourage different knowledge companies from subtly "enhancing" the KP protocols or data in ways that make it harder for users to concurrently use the servers of other knowledge providers? Indeed, marketplace or no, it makes no sense for there to be thousands of TPs that claim to reason about the same region of the network, so there needs to be some mechanism that manages the assignment of AS regions to TPs. (In one approach, a helpful provider can bring up a new node that offers to be a TP in the KP, but this algorithm, not the provider, then maps AS regions to the running servers.)

More broadly, this entire discussion tells us that the routing in the KP is not single-valued. Messages need to flow to more than one of the TPs that represent an AS – and how a message flows may depend on who asks it. And this raises questions of consistency (as different applications consult different TPs about the same problem) and reasoning in the face of inconsistency (as applications try to reconcile divergent answers).

Reasoning and Learning

Knowledge applications must be able to reason about themselves and their environment, respond robustly to surprise, and learn over time to become more effective. These requirements are implicit in the idea that the knowledge plane is at a higher level of abstraction than existing management tools. At this higher level it becomes necessary to take into account the evolving nature of network applications, their

interrelationships, dependencies, failure modes, and potential attacks.

Reasoning and learning also seem likely to play a key role in implementing the knowledge plane efficiently. For instance, consider the problem of the news of a network outage (e.g. a tunnel fire). Over time, the knowledge plane should learn what kind of information about an outage is typically accessed and which knowledge applications are most likely to want a particular piece of information. This learning can then drive how the knowledge plane circulates information about the outage, what knowledge it circulates, and how that knowledge is aggregated and dimension-reduced.

Here are illustrations of selected issues in reasoning and learning as they relate to the knowledge plane.

How can applications learn? Consider the *why* program. At an elementary level, when a new sort of failure occurs (one for which there are no explanations in the knowledge plane) people and organizations should be able to offer candidate explanations that are incorporated into the "memory" of the *why* program. Moreover, when a new application with new dependencies is fielded, the *why* program will need to be tutored about these dependencies and the normal behaviors of the program. Thus the heart of the *why* program must be a set of carefully structured models of network applications, and these models must be extensible or trainable in some form if the application is to learn.

How can learning be automated? Ideally, a mechanism for learning would not necessarily rely on human input. For instance, the *why* program ought to be capable of recognizing new anomalies, of generating and investigating candidate explanations on its own, of inferring new relationships between elements of a distributed application, and of proposing new diagnostic strategies. Results from theorem proving, planning, and expert systems may come into play to achieve this goal, and it is likely that the specifics of the networking context will also drive new research on automated learning if we are to meet extreme challenges such as recognizing and reacting to new worms in better-than-human time.

How can learning be validated? Of course, new explanations and strategies need to be validated in some way to weed out an incorrect or even maliciously inserted recommendation. One way to make progress safely would be to test a new hypothesis before relying on it. For instance, given a new set of policies that define a "working network", a configuration manager K-app could send trace packets to test the configuration it has put into place and see if packets actually flow the way they are supposed to.

Another progress mechanism could be collaborative filtering. Whenever the knowledge plane gives an explanation to a user or an operator, it could ask that person to rate the explanation. Explanations that get poor ratings could be weeded out. A refinement of the idea would be to develop mechanisms for selecting amongst alternative explanations. A source of prior highly-rated explanations would be a preferable source for some new information than an unknown or poorly rated source[3].

The relationship between models and learning: Bayes nets [4,5] illustrate the relationship between models and learning. An application based on Bayes nets exhibits a separation of concerns that supports three independent forms of evolution: of the engine, of a qualitative model, and of some quantitative parameters. Moreover, the qualitative aspects can either be "trained" or assigned. Another example is provided by the DIDUCE, a Java debugging environment that automatically infers and uses a model of correct program behavior [17]. DIDUCE works because the model consists of a set of invariants that can be adjusted automatically through training. Note that learning in DIDUCE is of a different kind than is supported by Bayes nets. What is common across these examples is the centrality of the model and its structure, which combine to enable learning.

Challenges: A challenge for the network community is to develop more useful models of the network, its users, and its applications and to incorporate these models into applications. A challenge for researchers in AI and Machine Learning is to develop new paradigms for structuring and working with these models in order to support performance areas such as network fault isolation and knowledge routing. Moreover, there will need to be a focus on automatic learning in an on-line setting, as opposed for instance to training for classification in off-line settings. Existing paradigms such as Bayes nets are illustrative and indeed directly helpful, but it is likely that new paradigms will need to be developed, and in the best case these will generalize to other settings

Building Blocks

There is a lot of basic research that is relevant to creating the knowledge plane. For instance, techniques such as epidemic algorithms [3] for distributing data, Bayesian networks for learning, and rank aggregation to enable a web of trust [6] are likely to be important in the development of the KP.

[3] Similar techniques are currently used to rank the relevance and quality of web pages [6,8].

However, direct experience in building a large distributed data management system with embedded cognitive abilities is limited. In the late 1980s, the DARPA-sponsored Automated Network Management (ANM) project sought to build a network-wide MIB collector combined with AI tools [7]. The ANM experience was that collecting data was relatively easy, but getting the data to the right place was hard – it was easy to overwhelm links with management traffic if information was circulated too aggressively. In a more limited scope, there's been considerable success using cognition to detect and repair localized faults in specific networked applications (e.g., HP's JetAdmin tool for managing networked printers and, much earlier, Greenberg's *why* program for Multics crash dumps) which might provide insight into building a network *why* tool.

Summary

This paper proposed to augment a network with a data plane similar to the Internet with a new higher-level artifact—a distributed system that addresses issues of "knowing what is going on" in the network. At an abstract level, this is a system for gathering *observations, constraints* and *assertions,* and applying rules to these to generate *observations* and *responses.* At the physical level, this is a system build out of *think points* that run on hosts and servers within the network. It is a loosely coupled distributed system of global scope.

As with many systems, the driver for the design will be the applications that run on it—in this case K-apps. A first step in architecting the knowledge plane is to look at existing research in modeling, analysis and control, and imagine how those tools might fit together synergistically. What requirements would they impose on the "five Rs" (representation, routing, reasoning, rating and response) of the knowledge plane? In parallel with this analysis, issues of scale and security must be considered; the knowledge plane is an opportunity to design a large distributed system in which there is a model of security built in from the beginning.

References

1. D.D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *Proc. ACM SIGCOMM '88*, pp. 102-111.
2. D.S. Eisenberg, "The Rise of the Stupid Network," *Computer Telephony*, Aug 1997, pp. 16-26.
3. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry, "Epidemic Algorithms for Replicated Database Management," *Proc. ACM PODC '87*, pp. 1-12.
4. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.
5. T. Bayes, "An Essay towards solving a Problem in the Doctrine of Chances," *Philosophical Trans. Royal Society of London* 53 (1763), pp. 370-418.
6. C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the Web," *Proc. 10th Intl. Conference on World Wide Web*, pp. 613-622 (2001).
7. J. Wescott, *Automated Network Management*, BBN Report No. 5641. BBN Technologies (1984).
8. L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*. Stanford Digital Library Project (1998).
9. V. Paxson, J. Mahdavi, A. Adams and M. Mathis, "An Architecture for Large Scale Internet Measurement," *IEEE Communications Magazine* 36 (1998), pp. 48-54.
10. J. Kleinberg, "The small-world phenomena: an algorithmic perspective," *Proc. 32nd ACM Symp. Theory of Computing* (2000), pp. 163-170.
11. L. McKnight and J. Bailey, ed. *Internet Economics*. MIT Press (1997).
12. D.D. Clark, J. Wroclawski, K.R. Sollins, and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet," *Proc. ACM SIGCOMM 2002*, pp. 347-356.
13. M. Mathis, "Diagnosing Internet Congestion with a Transport Layer Performance Tool," *Proc. INET '96*,
14. J. Padhye and S. Floyd, "Identifying the TCP Behavior of Web Servers," *Proc. ACM SIGCOMM 2001*.
15. V.N. Padmanabhan, L. Qiu and H.J. Wang, "Passive Network Tomography Using Bayesian Inference", *Proc. Internet Measurement Workshop 2002*.
16. R. Davis, H. Shrobe, and P. Szolovits, "What is a Knowledge Representation?" *AI Magazine*, 14(1):17-33 (1993).
17. S. Hangal and M. Lam, "Tracking down software bugs using automatic anomaly detection," *Proc. International Conference on Software Engineering '02*.