1. TITLE SLIDE - From Barnstorming to Boeing – Transforming the Internet Into a Lifeline Utility

Good morning, I’m Karl Auerbach.

My talk today regards what I believe is the necessary transformation of the Internet into a lifeline grade utility.

This is, I hope, going to be a short talk – I want to leave time for questions.

And, by the way, this obligates you to ask questions.

2. The Internet As A Utility

I suspect that none of us would disagree that the internet is becoming more and more a part of everyday life.

But are we *really* ready to really depend on the internet?

I don’t think so.

Most of our applications can handle intermittent connectivity and packet loss.

But near real-time systems will not be so forgiving.

Voice and security applications are particularly demanding – they require a solid, reliable network.

A burning building is not going to take time off if the fire alarm can’t get its packets through to the alarm company.

Even now the internet is being used for everything from security cameras to remote surgery.

We have every reason to expect that in the future the net will be used for even more systems that affect our health and safety.

As many of you know, I have been involved in a controversial organization that has as its goal the “stability” of the Internet.

This has led me to wonder what “stability” means and why we want it.

And this has required me to define what I mean by the word “internet”.

To me, the Internet is the open system of networks that supports and permits the unimpeded end-to-end flow of internet protocol packets between computer interfaces identified by IP addresses.

And on that Internet, the term stability means that these IP packets move with sufficient dispatch and reliability that we can build viable higher-level network applications.

I’ve used the term “lifeline utility” to emphasize that the fundamental services of the internet need to be so solid and dependable that we can entrust our personal health and safety to the operation of the net.

One of my pet fears is that someday there will be a real-life enactment of the old telephone ad in which an old person falls on his stairs, grabs his voice-over-IP wireless phone, dials 911, and gets a message saying that the call can not be placed because something like the domain name system or internet routing is out to lunch.

We can learn a lot from other disciplines – the history of railroads, airplanes, power grids, and telephony all contain lessons about building systems that must not fail.

Like these other utilities, the internet brings complexity beyond easy human comprehension and interdependency with other utilities.

But the internet goes further – on the internet there is a casual, even hostile, linkage between the parts coupled with a historical antipathy against outside control.

There are not many good things to say about the present state of the economy – but here is one: We happen to be in an era of excess capacity.

Because traffic is low, packets flow across the internet with few delays and relatively few points of congestion.

These halcyon days will not last forever; products that work well today may not do so well in the future when network conditions are not so favorable.

So let me restate my basic question: How are we going to turn the internet into a lifeline utility.

I'll begin by taking a look at our engineering practices.

3. Improving Our Engineering Practices

Let me be blunt: Our existing engineering practices are *<u>not</u>* going to create a utility grade internet.

Many of us here have worked on internet products, we have seen the pressures to get products working and shipped.

The focus is on getting products out the door and generating revenue.

Sometimes making a product robust isn't even on the agenda – that's usually left for the "next release".

And with the economic downturn, the demand for immediate financial returns from development investments are greater than ever.

As engineers, underdeveloped code is never our goal, it is never acceptable.

But as human beings we are fallible and unless the necessary institutional procedures are in place, under engineered products *will* reach customers.

OK, how do we build networks like Boeing builds airplanes?

4. Testing

Well, one obvious technique is testing.

Now, I mean something more than merely seeing if the thing works under normal conditions; I mean something rather more intense.

Consider automobiles.

Automobile builders in the US are not allowed to sell a new model until several cars have been reduced to scrap in a series of violent test crashes.

Critical internet software ought to be subjected to a similar battery of tests before it is loosed onto the public.

Adequate testing is something that customers ought to demand and we, as engineers, ought to support.

And we should retest every time code is changed.

Sure, this is motherhood and apple pie and everybody claims to be doing testing.

But testing is often under-budgeted and is frequently cut short by the demands of rapid product cycles.

And good testing is hard to do – test staff often needs to know more about the totality of a product than do many of the engineers who created it.

And few people aspire to be testers – the work is often boring and unrewarded.

Much of our testing consists of hooking up two boxes and seeing whether they interoperate.

Even under the best of conditions this kind of thing is a pretty sorry excuse for testing.

And to make things worse, a great deal of our network code comes from only a few sources.

Thus we often test against implementations that use the same algorithms, have made the same assumptions, and have done the same subsets of the protocols.

The result is code that is vulnerable to failures that occur when genetically different software is introduced into the net.

Some people take this situation seriously – I know someone who has tried to resurrect the MIT Incompatible Time Sharing System in order to find a TCP stack that shares virtually no code DNA with existing implementations.

Because of the lack of wide code diversity test suites are very important.

Test suites are intentionally designed to systematically explore protocol options and the corner cases of algorithms.

I have recently built a new product to help with the testing problem.

I call it “Maxwell” after the daemon posited by James Clerk Maxwell in 1871.

This product allows the user to introduce controlled perturbations into network protocol flows in order to exercise the otherwise under tested parts of network implementations.

Effective bug reporting

Most of us think of testing as something that happens in the Q/A lab and in the hands of customers who know they are participating in alpha or beta tests.

However, in real life, much testing occurs as products are put to actual use by unsuspecting customers.

I am sure that every one here has encountered a bug and has discovered no way to report it or to get it fixed.

Vendors must improve their support mechanisms so that users’ bad experiences aren’t endured in vain – there should be effective and easy means for users to report bugs.

5. Design Rules

Most mature engineering disciplines make heavy use of design rules.

Design rules are a way we pay tribute to the mistakes made by those who came before.

Some of these rules, such as building codes, have even been given the force of law.

For some reason, in software and in networking, we try to be independent and reinvent the wheel again and again.

That is something we need to change.

On the slide I've listed four examples of useful design rules.

I hope that most of these are fairly obvious.

However, I want to give special mention to the idea of using protocol frameworks

Consider BEEP, RFC3080.

Rather than inventing arcane transports for every different application, BEEP gives us a nice flexible, pre-thought-out structure.

BEEP lets us stand on the shoulders of giants so that we can build new and useful things more quickly and with fewer flaws.

But we need to be careful, as we have seen in Microsoft Windows, blind use of frameworks can lead to bloated code and slow performance.

While I am on the topic of efficiency - Efficiency is rarely our primary goal, we can afford to sacrifice bytes-on-the-wire and CPU cycles in order to gain reliability and safety.

I do want to point out that too many of today's software and network engineers do not fully comprehend the totality of their work.

Too few of us really understand how what we do at upper protocols layers really operates down at the level of bits on the wire.

This can lead to disaster.

In Stockholm there is a museum built around a single ship, the Vasa.

The Swedish engineers of the early 1620's built a huge warship with too many layers, too many canons, and too little understanding of weight and balance.

On the Vasa's first trip away from the docks, a light breeze tipped it over and it sank.

We must be careful to fully understand what we are building, otherwise we may build network Vasa's that will sink with our customers on board.

Let me finish my thoughts on improving our engineering practices by mentioning something unpleasant…

6. Legal Liability For Flaws

Not everyone suspects this, but I am a card-carrying lawyer.

Yup, I'm one of *those* people.

Being a member of the evil race, it seems natural to me to suggest that we might improve engineering by raising the stakes a bit.

And I'm not alone…

Last fall, at the IETF meeting in Atlanta, Bruce Schneier suggested that one way to address network security problems was to impose legal liability upon those who are negligent or reckless with regard to the security of their products.

Such liability is in-line with traditional product liability laws, laws that sometimes hold product vendors to the ultimate standard of strict liability for flaws in their products.

I personally believe that liability for flaws is a good thing.

However, laws such as UCITA are moving in the opposite direction – these laws allow software vendors to repudiate their responsibility for the improper behavior of their products.

By the way, it is quite common for states to deny insurance protection for certain kinds of acts, the idea being that some things are of such importance that people ought not be able to forget their responsibilities simply by buying an insurance policy.

7. Changing Our Engineering Conceptions

Now for the more fun part of the talk – here's where I'll wander out into the great blue unknown and suggest that we take another look at the net to see if we can approach it in new and different ways.

I will, and do, assert that merely improving code quality isn't going to get us to the Network Nirvana.

We gotta do more –

We need to change some of the ways we think about networking.

Some of these changes are small, some huge.

Some may ultimately prove to be worthless. But that does not refute my assertion that we have to look at the net in new ways.

OK, let's look at my list…

8. Engineering Conceptions

Let me give you a moment to skim the list on the slide.

But only skim it – I will deal with each point separately in a bit.

In the meantime, I’d like to ask you to consider what you would add to the list.

9. Fail-Safe Design

I am a railroad nut. So I was thrilled when Santa Clara County built a major streetcar junction right outside my window at Cisco.

The engineers who designed and built the rail switch clearly had a worldview quite different than what is common in the networking businesses.

The railroad folks built the switch out of the highest quality parts, it was so overbuilt that there was no chance it could ever break.

But it was also clear that they looked at it and asked themselves "but what if it *does* break?"

So they added additional parts to make sure that failures would be benign.

And they added multiple additional parts to detect and indicate inconsistent movement of the switch.

All in all it was very impressive.

Which reminds me of a story – Not many people know why red means stop and green means go.

Well, when railroads began they used white signal lights to let the engineer know when it was safe to precede.

When there was danger ahead, they raised a red lens in front of the light.

Thus white meant proceed and red meant stop.

This worked just fine….

…that is until the red lens fell out.

So they changed the system so that a separate green lens was used to indicate that it was safe to proceed.

Then, if a lens fell out and you saw white light you knew that something was amiss.

That is fail-safe engineering.

10. Distinguish Network Management From Troubleshooting

One of my pet peeves is that we have largely failed to distinguish network management from troubleshooting.

The SNMP protocol was crippled from birth by this failure.

I would like to suggest that network troubleshooting be considered a discipline distinct and different from network management.

Network management can depend on most of the infrastructure of the net being available; troubleshooting, on the other hand, must have minimal dependence on outside mechanisms.

There is a lot of room for network management and troubleshooting tools to complement one another – for example network management can provide troubleshooting tools with a schematic for what the net ought to look like, and troubleshooting tools can tell network management what the net actually looks like.

The point I want to make here is that our chances of transforming the net into a lifeline utility will be increased if we clearly define understand our focus.

11. The Net As A Distributed Process

On the internet no device is an island.

Yet many of our network problems are caused not by the failure of individual boxes but rather by missteps in the choreography of the interactions between them.

A good way to manage the net is to use process control principles.

We ought to think in terms of closed feedback loops and hysteresis.

Many, perhaps most, of the existing control systems on the net are undamped open control loops.

That’s a technobabble way of saying that we have a house of cards.

Network management systems ought to incorporate feedback loops so that the application of a control pressure – for example the grant of priority QoS to a flow – is resisted by a measure of the impact of that flow.

Automatic responses, such as triggered scripts should be damped so that we don’t get into a rapid-fire click-on, click-off kind of situation.

This kind of process control thinking becomes more important as we make the net more able to take autonomous action.

And thus we come to the next item…

12. The Not-So-Dumb Network

I hope that we have all read David Isenberg's wonderful paper the *Rise of the Stupid Network*.

It's a great paper and full of wisdom.

And I agree with it in general.

But …

I have always had some trouble with the idea that the net should be utterly devoid of intelligence.

Sure, I agree that the packet-forwarding plane of the network ought to be streamlined and have minimal awareness of packet contents.

But on the other hand, I don't agree that the systems that *control* the data plane should themselves be simpleminded.

Nor do I agree that devices shouldn't look over their own shoulders to see whether they are doing a reasonable job.

In a project I worked on about two years ago I was trying to develop the notion of configuring routers based on goals rather than specific settings of specific configuration items.

That system required that the router have a degree of autonomy to adapt the details of its configuration in order to meet the goal.

This did not require any changes to the data plane.

The data plane remained utterly stupid.

But it did require me to consider a router to be a full fledged computer that could run programs that examined the router's behavior, compared that behavior to a set of goals, and made adjustments as necessary.

So, let's not go overboard on the notion of the dumb network - devices *are* smart enough to participate in their own management.

13. Competing Algorithms

Many systems are most stable when that stability is the result of two or more competing mechanisms that pull against one another.

I'm not necessarily talking about complicated things – a competing algorithm could be as simple as a timeout that takes down new configuration settings unless a message is received from an operator that the new settings are working.

Having locked myself out of a remote router on too many occasions, I can give a first-hand testimonial to the value of automatic fallback of configurations.

I also want you to understand that I'm not talking about optimality, I'm talking about stability – stable systems are not always optimal.

An area in which I feel that competing algorithms will be particularly valuable is in the area of Quality of Service provisioning at the edges of the internet.

The job of adapting to requirements of service level agreements, particularly with demanding applications such as conversational voice or video, will quickly go beyond the ability of human operators.

And there may simply be no time to compute and recompute optimal solutions.

One answer to a given problem may be to adopt a single algorithm that tries to come up with a usable result.

However, a more stable outcome might be obtained by using *two* algorithms, one that is trying to grant better service levels and another that is trying to reduce them.

The result is a dynamic tension that hopefully leads to a usable, if not optimal, configuration.

14. Management by Delegation

Several times I have mentioned the idea of delegating limited autonomy to devices.

In human enterprises, management is done by delegating authority.

This is a good model to follow in networks.

Because of the time delays involved in coordinating with a central management authority, management by delegation gives us the ability to have a “man on the spot” able to take quick action.

This can be done by hard-coding procedures into the box.

Or it can be done by means of dynamic delegation.

In its most basic form this is simply dynamic scripting – sending scripts into a box for execution.

Scripts really encompass four things:

- Procedures to be followed
- The set of resources to be managed
- Limiting constraints
- Desired goals

I’ve found this structure to useful, particularly when I consider the last three items to be parameters to the first.

I am, of course, completely ignoring the extremely hard issues of security and of encapsulating the scripts to limit the harm that they might cause.

By its very nature, network management has to have the ability to cause harm to the network.

Network management ***is*** a dangerous instrument, and like a surgeon’s scalpel, we really do want it to be sharp.

15. Self-Healing Networks

I have been an advocate of self-healing techniques for more than a decade.

I'm glad that it is finally becoming a topic for serious discussion.

It is something that we need to do, but we need to be extremely cautious – we are likely to find that the mechanisms that are supposed to heal turn out to be rather effective means to quickly turn a net into a non-functioning heap of routers, switches, and servers.

Any time a network is given the freedom to adapt and configure itself we risk losing track of the network's configuration and logical topology.

This loss of knowledge of the configuration could have a negative effect on the effectiveness of tools such as event correlation or intrusion detection.

My conception of self-healing does not include robots that go out and repair broken electronics or splice fibers torn apart by a passing backhoe.

Rather, when I think of self-healing, I tend to think of "healing" as adjusting configuration parameters.

In that sense, self-healing is a kind of auto-configuration mechanism that tries to find workable solutions to mis-configurations.

Self-healing as a general proposition is too large a topic for us to consider, much less to solve.

However, I believe that we can start building self-healing mechanisms today if we pick constrained problems.

Quality of service provisioning seems to be about the right size.

So how do we get started?

16. Self-Healing Networks – Taking the First Steps

How do we actually build self-healing networks?

We get started by thinking small.

We find a bounded issue, preferably one that isn't critical to the fundamental service of the internet, the transport of packets from hither to yon.

That way, packets will still flow even if we mess up – and I guarantee that we ***will*** mess up.

And we ought to begin my merely mimicking the procedures used by human operators and allow the configuration of the net to be changed only after obtaining permission from a human.

The self-healing mechanisms ought to be designed using all the principles described previously – feedback loops, auto-fallback to working states, competing algorithms, and management by delegation.

The time seems ripe for us to cautiously start to build limited self-healing mechanisms into our networks.

17. Study Network Pathology

The best way to describe our approach to network management is *ad hoc*.

And we will continue to be ad hoc until we have a better knowledge of the relationship between network causes and network effects.

We need to study network pathology.

We need to get more serious about building causality databases and models.

Network failures and errors, and their causes, need to be recorded and analyzed.

Yes, good work has been done on this, however compared to other disciplines we are still in the stone age.

18. Management and Reparability From The Outset

There are still too many product designers and vendors who consider management something to be glued on rather than built in.

As the net moves towards being a utility, and as network products achieve commodity status, vendors will not survive unless they can control their support and repair costs.

And that means that devices must become more manageable and more repairable.

We can learn a valuable lesson from the automobile industry.

Back in the 1970's auto manufacturers started to use electronic control systems and began add diagnostic plugs to cars.

Those primitive mechanisms have greatly evolved since then.

Many of today's hot automotive features – variable ignition and valve timing, adaptive performance settings, and self-adjusting tuning, to name only a few – are direct outgrowths of the management interfaces built into cars 25 years ago.

To say it more clearly – the automobile industry discovered that automotive diagnostic and control systems, in other words, management systems, turned into a means to provide the customer with a rich set of new performance features.

The lesson that we should draw is this – network management, rather than being a stepchild technology, is actually a very important way to add new features to products.

19. Conclusion

So what do I conclude from all of this?

Mainly that there is a lot that we can do to make networks better.

And this work can be intellectually stimulating.

In fact, network management represents the convergence of many technical disciplines.

Network management can be exciting – and profitable.

And now for your questions and comments…

20. Closing Slide