Failure to Thrive: QoS and the Culture of Operational Networking

Gregory Bell
LBLnet Services Group
Lawrence Berkeley National Laboratory

ACM / SIGCOMM - 27 August 2003

Introduction

- I'm a network engineer at LBNL
  - not a researcher; not a protocol designer
  - recent experience with IP multicast
- I'm here to explain why we have not deployed QoS
- And more generally, to argue that a reasonably-rich version of QoS may not be deployable

What is Quality of Service?

- A set of architectures & technologies that provide
  - an alternative to best-effort packet delivery
  - preferential treatment for certain traffic flows
- A technique for meeting the needs of delay- and loss-intolerant applications, e.g.:
  - voice over IP (VOIP)
  - video-conferencing
  - real-time gaming
  - online surgery?
- So far, not a roaring success

Why is this failure noteworthy?

- Stature of QoS architects
- Volume of QoS activity
  - dozens of articles, Internet Drafts, RFCs, dissertations, books
  - opportunity cost?
- Highlights a rift between protocol design and network operations
  - this rift has implications beyond QoS

Overview of my claims

- The culture of operational networking helps explain why QoS floundered
  - that culture is averse to complexity, and QoS is highly complex
- IP multicast is a useful lens
  - like QoS, it supplements best-effort unicast
  - defines a functional limit for deployable complexity
- Asking "*what is deployable?*" raises questions about economic, historical, institutional forces
  - often ignored in protocol design

The aversion to complexity

- Lots of recent work on complexity in large-scale networks
- A common refrain: the Internet is "robust yet fragile"
- Various explanations for the source of fragility: amplification, coupling, human error, hardware failure… what else?

Complexity underestimated

- But this scholarship underestimates the impact of design complexity on stability
- Assumes that frailty comes from the unintended consequences of well-behaved systems interacting
  - *e.g.*, synchronization of routing updates
- But complex protocols don’t always function as they were designed to function
- Failure is more likely to be caused by a software bug than by unexpected feature interaction

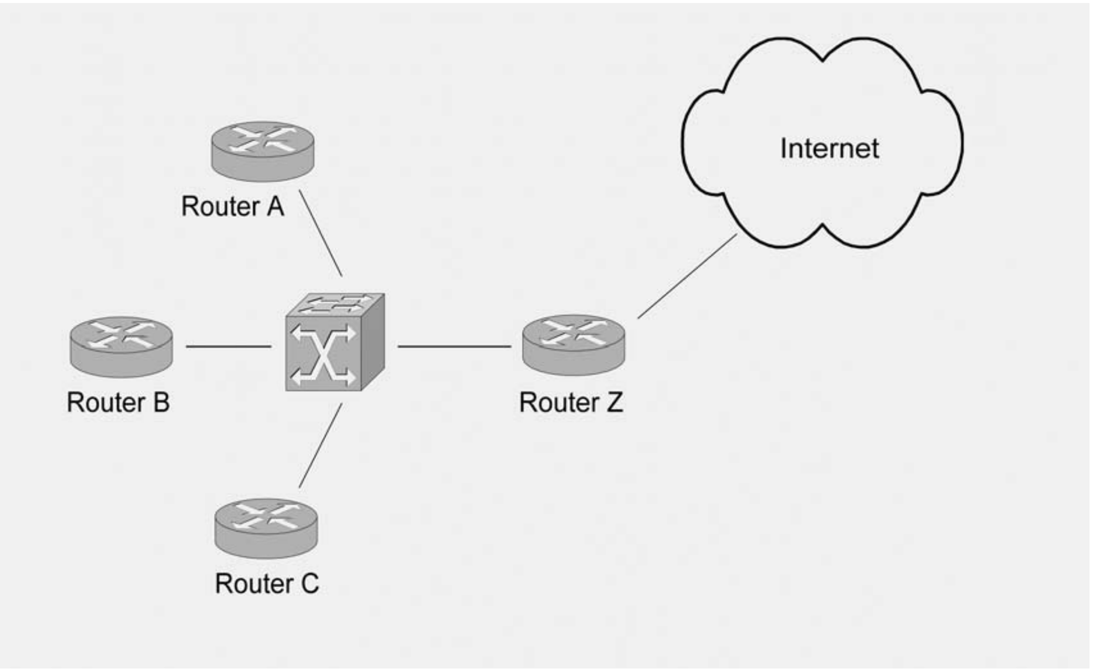
Impact of software bugs

- Complex protocols are sometimes implemented poorly in routers
  - especially when the constituency is small and the deployment modest (*eg*, MSDP)
- Working network engineers encounter serious anomalies on a regular basis
  - routers crash
  - interface buffers wedge
  - packet counters show negative values
  - advertised features don't work
  - implementations from different vendors don't interoperate

Impact of software bugs

- A recurring operational cycle: we debug, we upgrade, we test
- As a result, we anticipate and plan for failure
- Not simple pessimism; a form of working knowledge
- It’s difficult to appreciate this perspective without living through
  - many new deployments
  - the associated debugging sessions

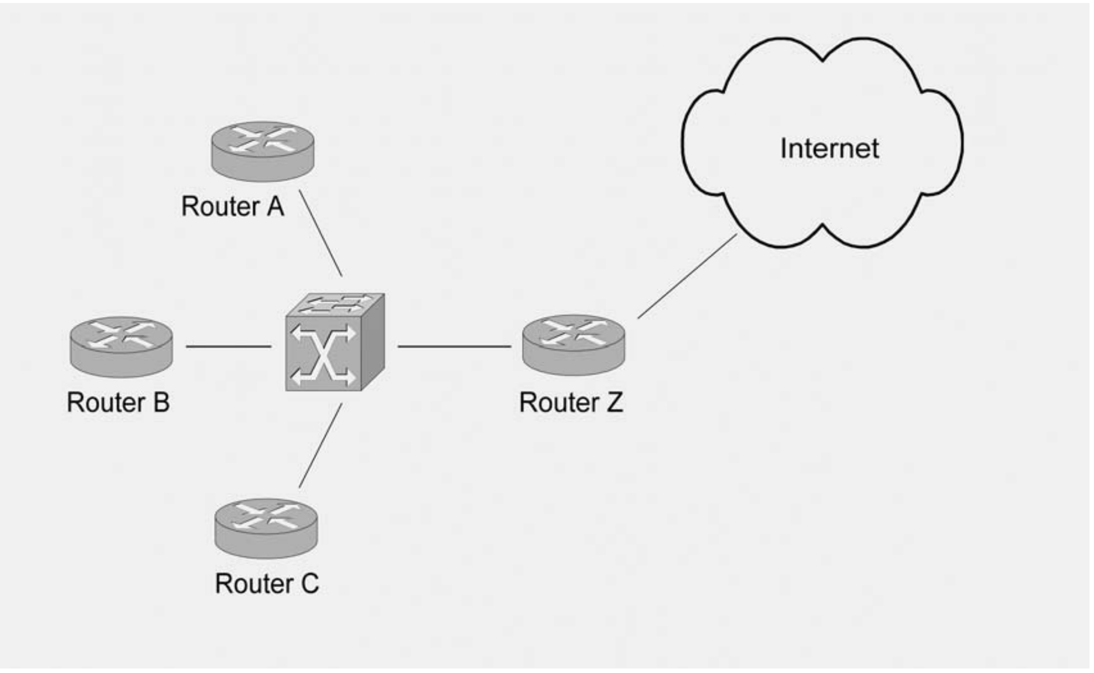
An example of failure



(simplified network diagram)

- One day, all subnets served by Router A lost connectivity with the outside world, followed by subnets on Router B, then Router C
- Internal connectivity was fine
- BGP and OSPF appeared normal

An example of failure



(simplified network diagram)

- We isolated the problem to a failed ARP process on Router Z
- When ARP cache entries on A, B and C timed out, each router stopped forwarding packets to Z
- The ARP failure was traced to a route processor crash triggered by a multicast bug

A pattern of failures

- This failure fit into a much larger pattern
- In the past year, we had coped with over a dozen major multicast bugs
  - affecting PIM, MSDP, IGMP, CGMP
  - on 5 different hardware platforms
  - almost all of them caused by software bugs (predominantly in the data plane, not the control plane)
  - one or two bugs related to interoperability
  - no failures related to misconfiguration
- Time required to debug everything was ~engineer-weeks

A pattern of failures

- Spectacular symptoms
  - router reboots when it sees normal multicast traffic
  - router reboots when setting up MSDP peering
  - buffers wedge with normal PIM and IGMP packets
- The bugs don't just affect multicast performance – they hurt the stability of unicast routing

Deployability

- Our “multicast meltdown” is relevant to the fate of QoS
- IP multicast defines a likely functional limit for deployable complexity
- This does not mean that multicast (or QoS) is “too complex” to be implemented reliably

Deployability

- The issue is whether it can be implemented reliably given the factors that constrain the success of real-world deployments, including a lack of:
  - adequate quality assurance by vendors
  - critical mass of customers
  - debugging tools
  - knowledge in the enterprise
  - trust between neighboring domains
  - a business case to justify correcting the other problems

Implications for QoS?

- To deploy QoS is to confront most of the real-world constraints encountered with IP multicast
- Intuitively it's clear that QoS can be just as complex as IP multicast, and potentially more so
- Of course, complexity varies according to the flavor of QoS

Integrated Services (IntServ)

- The clearest case
- Routers (even core routers) keep per-flow state
- Reservation setup is “fundamentally designed for a multicast environment” [RFC 1633]
- Take the complexity of inter-domain multicast, then add reservation setup, admission control, classification, packet scheduling, and more
- Never widely deployed

Differentiated Services (DiffServ)

- **This is the live issue**
- **Complexity of DiffServ harder to assess, thanks largely to its flexibility**
  - **aims to be scalable by aggregating traffic classification through IP-layer marking**
  - **“agnostic about signaling”**

Minimalist DiffServ

- DiffServ can be implemented on a modest scale, maybe a single bottleneck
  - only one router in a network pays attention to DiffServ marking
  - let's call this model "minimalist DiffServ"
- Minimalist DiffServ is a far cry from Grand Unified QoS (as exemplified by IntServ)
- But can it really provide the rich service model envisioned by QoS architects and advocates?

Slightly-less-minimalist DiffServ

- Reasonable utility → increased complexity
- For instance, it might be nice to:
  - enforce a policy more nuanced than “VOIP traffic gets precedence”
  - enlarge the diameter of the DiffServ domain to include several routers, an entire network, a collection of networks
  - harden DiffServ against DOS attacks and resource theft
  - implement protocols for resource availability discovery, service requests, provisioning, dynamic traffic engineering
  - provide auditing, tracking and debugging information

QoS and complexity

- The big question: are *useful* models of QoS *deployable*?
- Remember all the constraints in the multicast case:
  - adequate quality assurance by vendors
  - critical mass of customers
  - debugging tools
  - knowledge in the enterprise
  - trust between neighboring domains
  - a business case to justify correcting the other problems

Thinking like a network engineer

- To ask "is this deployable?" is to start thinking like a network engineer
- Among other things, that means considering:
  - price of router interfaces
  - price of wide-area bandwidth
  - current incidence of latency, jitter, packet loss
  - customer demand for real-time applications
  - skills of engineering staff
  - time-to-resolution for complex problems

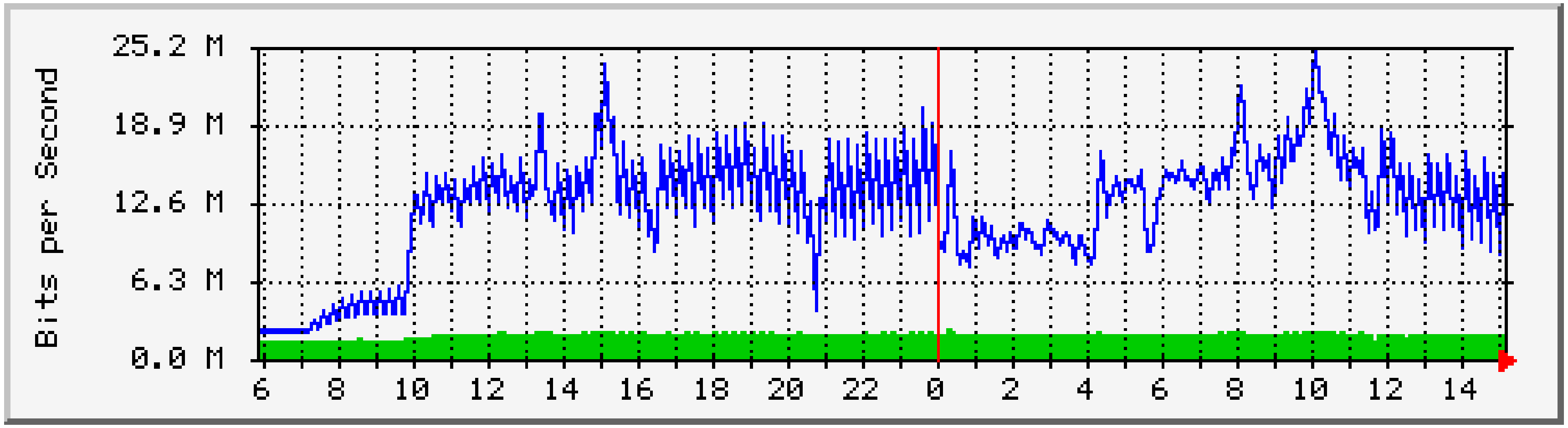
Thinking like a network engineer

- It means asking very pragmatic questions when evaluating a new technology:
  - what does my network have to gain from enabling this?
  - is the necessary test equipment affordable?
  - can I debug it w/o impairing best-effort service?
  - when debugging, do I need active cooperation of engineers in other domains?
  - are the benefits sufficiently compelling to compensate for potential pain?
  - when it breaks, will I be blamed?

Thinking like a network engineer

- **And more:**
  - **am I likely to be caught in the middle of disputes regarding who gets premium service?**
  - **will I be asked to investigate very transient, vaguely-defined symptoms that users attribute to the failure of QoS?**
  - **will QoS become a black hole for my time, and that of my colleagues?**
  - **isn’t there an easier way?**

Throwing Bandwidth



5-minute average load on internal GigE router interface

Throwing Bandwidth

- **This is the primary operational response to jitter, latency, packet loss**
- **But the formulation is misleading (sounds un-engineered, *ad hoc*, "inefficient")**
- **Throwing bandwidth has more merit and more staying-power than some QoS advocates have been willing to acknowledge**

The “10% rule” at LBNL

- When average utilization of router interface exceeds 10% of link speed, upgrade
- We assume our monitoring systems don’t tell us much about transient, peak utilization
- It’s simple, and it works well in practice
- Is it economical?
  – that depends on the market for Ethernet interfaces (especially router interfaces) when the 10% boundary is crossed
  – in practice, the rule has not committed us to bleeding edge
  – current cost to upgrade from 100 Mbps to 1Gig subnet feed, in our environment: ~ $1500 US

The “10% rule” at LBNL

- When all costs are carefully considered
  - we think that throwing protocols at the problem will compromise stability
  - and throwing bandwidth is the cheapest antidote to congestion on our network

Throwing Bandwidth

- Researchers have begun to explore “over-provisioning” as a possible alternative to QoS
  – one study shows that at high link speeds, the excess capacity required to minimize latency is only 15% above average utilization
  – operators are reporting similar results
- But economic context makes all the difference
  – a dramatic change in the market for bandwidth, or the demand for it, might make this strategy less attractive

Conclusions

- Remarkable intelligence and energy have been lavished on the design of QoS
- Much less attention has been devoted to a careful analysis of the relevant problem space from an operational or economic perspective
- This discrepancy is symptomatic of a broken feedback loop between network operations and research
- Ideally, there would be a constant exchange of information between these domains

Conclusions

- In practice, research and operations are mutually-insular
- Few people / institutions are able to bridge the gulf
- This rift has harmed the process of protocol design by shielding it from the daily experience of failure in enterprise networks
- Such experience is important in estimating the limits of deployability
- Until the architecture of QoS is calibrated with these limits in mind, it will continue to suffer from a failure to thrive

- Thank you!
- Contact info: grbell@lbl.gov
- Slides will be here:
  http://gravity.lbl.gov/grbell/
- I am grateful to Ted Sopher, Mike Bennett, Deb Agarwal, Jim Leighton, Ion Stoica, and Sally Floyd for helpful feedback on my paper and presentation