DNS and Internet Mail Service Architecture

Joe Abley <jabley@isc.org>

INTERNET SOFTWARE CONSORTIUM

Agenda

- General Service Architecture Design Process
  - Application to DNS
  - Application to Internet Mail
- Brief tutorial overview of DNS and SMTP
- Lots of case studies and commentary from you (please interrupt frequently)

These Slides

`http://www.isc.org/misc/netsa2003/dns-and-mail.pdf`

- You might like to take notes
- These slides will not be a good record of my handwaving, my elaborate whiteboard scribbling or of the useful experience you hear from other people in the room

Service Architecture Design

What is a Service?

A Service has...

- a particular job to do
- names or addresses (or both)
- a set of clients
- dependants and dependencies
- availability requirements
- a tendency to become busier
- security requirements

Service Function

- It is useful to know what your service actually does before you think about deploying it
- if you don’t know what it is supposed to do, you don’t know whether it is doing its job

Service Location

- How do clients locate the service?
  - DNS name?
  - IP address?
  - Some other method?
- Why do we care?

Identification of Clients

- What are the clients who will use this service?
- Do I know who they are?
- Can I send them all mail to tell them stuff, if I need to?
- What will they do if things break?

Dependencies

- What other services does this service depend on, in order to function?
- What other services depend on this service?
- Dependencies limit availability

Availability

- A service is available if it seems to function correctly from the perspective of a client
- What are the hours during which we need to ensure the service is available?
  - 9am - 5pm, Monday to Friday?
  - 24 x 7 x 365?
- Do we get maintenance windows?

Growth

- If the service is any good, the chances are its workload will increase
- We need to be prepared to increase the performance capability of the service as it grows
- We need to be able to measure service performance so we know when we need to scale it up

Security

- We need to be prepared:
  - to restrict the use of our service to the clients we intend to service
  - to deal with clients who are not behaving nicely
- Sometimes this will impose additional requirements on other services on which our new service depends

DNS

6-Slide DNS Tutorial

DNS Tutorial 1

- The DNS provides a mechanism for mapping names to resources
- The DNS consists (principally) of:
  - a namespace
  - resource records
  - nameservers

DNS Tutorial II

- The namespace is a tree of labels descended from a common root
- The namespace consists of zones and domains
- Zones are connected by delegations
- Delegations are all about authority and nameservers

DNS Tutorial III

- There is usually a single (”master”) source of authoritative zone data
- That zone data is distributed to other authoritative servers (”slaves”) using mechanisms such as “zone transfer”
- The timers which control zone transfers are specified in the zone data

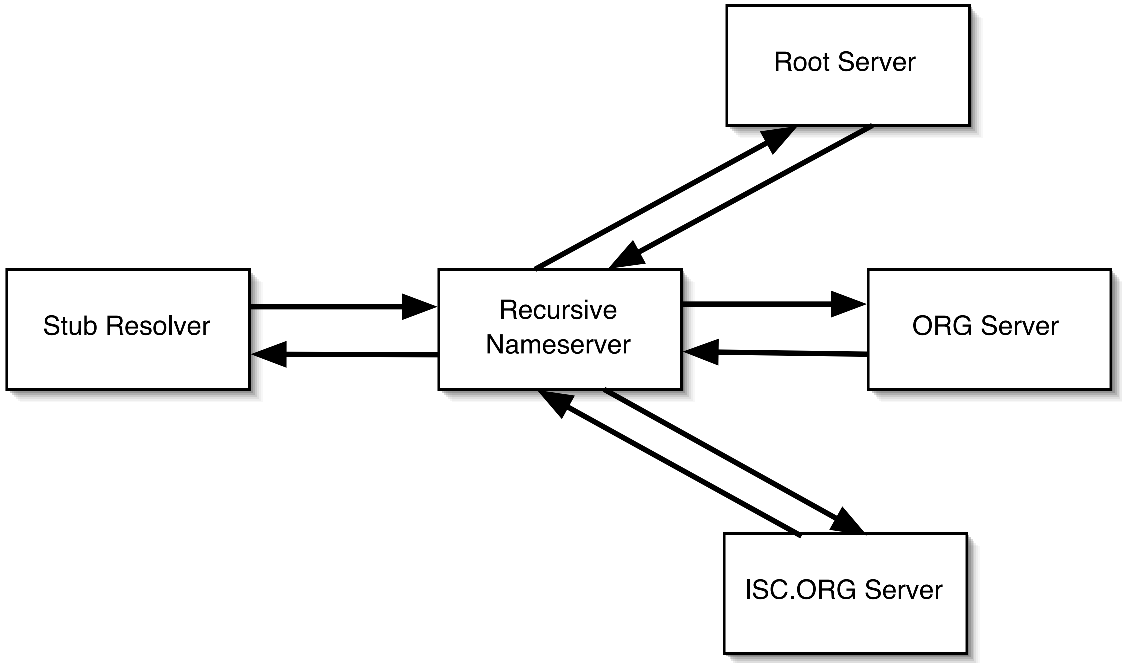
DNS Tutorial IV

- Resource records are stored in zone data
- Resource records are retrieved by resolvers sending queries to nameservers
- There are lots of different resource records
  - SOA, NS, A, AAAA, MX, SRV, ...
- Each resource record has a TTL specified in the zone data

DNS Tutorial V

- Applications typically talk to stub resolvers
- Stub resolvers typically talk to caching resolvers
- Caching resolvers talk to authoritative nameservers

DNS Tutorial VI



Service Separation

- Nameservers can provide multiple functions
  - authoritative nameservers
    - master server, slave server
  - caching resolver (dns cache)
- We are going to consider these functions as separate services
- Why?

Caching Resolver

Function

- Perform recursive lookups on behalf of stub resolvers
- Cache responses so that they can be returned to stub resolvers rapidly

Names, Addresses

- Service referred to by clients using one or more IP addresses
- Addresses can be handed out to clients using dynamic protocols (DHCP, PPP/IPCP)
- They can also be hard-coded, and we should expect this to happen
- Renumbering will probably be painful

Clients

- A well-known set of clients
  - “dial-up users”
  - “DHCP clients”
  - “workstations within our network”

Dependencies

- What must work in order for the Caching Resolver to work?
  - Network access to clients
  - Network access to external nameservers, or
  - Network access to one or more caches

Dependants

- What depends on the Caching Resolver service?
  - Browsing the Web, Downloading Stuff
  - Mail, Instant Messaging
  - Printing, File Service, maybe
  - Everything

Availability

- The Caching Resolver inherits all the availability requirements of all dependent services
- For an ISP (and most non-trivial enterprises) this means the service needs to be up all the time
- Maybe we get a maintenance window

Growth

- As the use of network services increases, so the load on the Caching Resolver service will increase
- For most enterprises this growth won’t require a very rapid increase in CPU or network
- For ISPs, the growth might be higher

Security

- The DNS is insecure in many ways
  - Details available in the DNS workshop
- There isn't really a good way to secure the data obtained from the DNS today, but maybe there will be some day
- We can restrict access to known clients

Data Gathering Over

- What are the implications for designing our service architecture?

Huge Availability

- We ideally want the Caching Resolver service to always be available, since so many things break when it’s not available
- Accidents will happen, but we’ll try to make ourselves invulnerable to as many of them as possible
  - helpdesk phone ring bad

Address Stability

- We don’t want to renumber our caching nameservers, ever, if we can help it
- Avoid topology-sensitive addresses
- Avoid unstable addresses, e.g. PA addresses allocated by ISPs
- Try very hard to hand out addresses dynamically wherever possible (DHCP, PPP/IPCP) so that we minimise the pain if renumbering is ever necessary

Service Addresses

- Give servers topologically-relevant addresses for management
- Give services topologically-independent addresses, which can be routed towards particular servers
- Allows you to move services between machines, and to re-plumb your network without having to renumber services

Danger!

- It is usual for servers to have a single IP address, and for routers not to require extra configuration to reach particular servers
- Adding complexity might cause operational confusion
- We don't want the measures we take to increase our uptime to cause network problems

Service Distribution

- We can provide service on a particular address in more than one place
- Local load balancing
  - (”layer-4 switches”, “content switches”)
- Anycast
  - using the local network’s routing system to distribute service load

Server Diversity

- We should deploy more than one server (or set of servers, if distributed)
- located in different places
- each with a unique, stable address
- each maybe running different DNS software

Single-Purpose Servers

- Caching Resolvers typically require less maintenance than other services, and don't place much demand on the underlying OS
- Deploy cheap servers which run the Caching Resolver service, and nothing else
- Cheap but Reliable!

Growth

- Add more servers (load-balancing switch, anycast, additional service addresses)
- Introduce a cache hierarchy

Measurement

- Individual nameservers will produce statistics which you can gather and use to trend performance data
- Test infrastructure
  - distributed, if using anycast
  - Caching Resolver failures should ideally be caught early, before too many other things start to break
  - more than just ping

Summary

- Reduce the possible impact of dependencies by designing around them
- Choose IP addresses wisely
- Use Service Addresses
- Service Distribution
- Cache Hierarchy
- Measure performance, so you know how you are doing

Authoritative Nameservers

Function

- Serve authoritative zone data to recursive resolvers
- Publish authoritative zone data in the DNS
- Delegate authority for child zones to other nameservers

Names, Addresses

- Service referred to by clients using one or more IP addresses
- Addresses are obtained by clients from other authoritative nameservers (nameservers for the parent zone)
- Renumbering will probably be painful (why?)

Clients

- A set of clients which we can’t enumerate
  - “every caching resolver in the Internet”

Dependencies

- What must work in order for the Authoritative Nameserver service to work?
  - Network access to clients (i.e. to the Internet)
  - Access to authoritative zone data

Dependants

- What depends on the Authoritative Nameserver service?
  - Every other service we want to make available which is referred to by name
  - Internet Mail
  - (downtime mitigated by external caches)

Availability

- Popularly-requested records will be cached externally, so our availability requirements are not as high as for the caching resolver service, maybe
- Things will break in strange, unidentifiable ways if the service is down for long
- Probably we can tolerate downtime for individual servers, for maintenance
- We should try for very high uptime across the NS set

Growth

- If we need to publish volatile data in the DNS, then our query load will go up (since fewer queries will be answered by caches)
- If we publish data which relates to services which are growing, our query load will go up
- If we host more zones, our query load will go up

Security

- We should be careful to provide our usual high levels of system security on authoritative nameservers (why?)
- One day, we might have to worry about DNSSEC
- We can’t enumerate our set of clients, so we can’t restrict access to them
- We are vulnerable to nameserver or protocol exploits (so patch early, patch often)

Data Gathering Over

- What are the implications for designing our service architecture?

Good Availability

- Our requirements for uptime are not as hellish as for the Caching Resolver service
- We should aim for very high uptime, and take steps to ensure that downtime of individual servers are tightly controlled
- Accidents will happen, but we'll try to make ourselves invulnerable to as many of them as possible

Address Stability

- We don’t particularly want to renumber our authoritative nameservers, but we can be fairly sure that people haven’t hard-coded them
  - Talking to registries can be tedious, and is best avoided if possible
- Avoid topology-sensitive addresses
- Avoid unstable addresses, e.g. PA addresses allocated by ISPs

Glue Records

- The DNS delegation tree depends on glue records for delegations to nameservers that are named in-zone
- Changing glue records can be difficult, particularly if they are used for high-profile delegations

Resolver Distance

- Delegating zones to nameservers which are named deep within some other hierarchy can cause delays and timeouts for resolvers
- Choosing at least some nameservers which are named in-zone (and are hence available via glue) is a good idea

Service Addresses

- Useful for the usual reasons:
  - Give servers topologically-relevant addresses for management
  - Give services topologically-independent addresses, which can be routed towards particular servers
  - Allows you to move services between machines, and to re-plumb your network without having to renumber services

Service Distribution

- We should provide service in more than one place, using multiple NS records
- High geographic dispersion
- The intention is to make the service highly available to the entire Internet, and we have minimal control over the majority of the Internet

Slave Servers

- There are commercial providers of slave DNS service
- There are lots of free providers of slave DNS service
  - (look around you)

Other Peoples' Slaves

- It can be an advantage that you don't personally run slave servers
  - if you don't have administrative access, you can't break them
- It can also be a disadvantage
  - getting changes made might take time
  - you are dependent on others to avoid vulnerabilities and mistakes

No Recursion

- You do not want to use slave servers which are also caching resolvers (why?)
  - Recursion should be turned off, completely
- You should check slave servers which are run by other people periodically, to make sure that recursion hasn’t been enabled

Growth

- Add more NS records: possibly painful (why?)
- Distribute service for each nameserver address, as we discussed with the caching resolver
    - (load-balancing switch, anycast, additional service addresses)
    - You can buy this

Measurement

- Individual nameservers will produce statistics which you can gather and use to trend performance data
- Test infrastructure
  - distributed, if using anycast
  - Authoritative nameserver failures should be caught early, before data expires from caches and slaves
  - more than just ping

Summary

- Reduce the possible impact of dependencies by designing around them
- Choose IP addresses wisely
- Use Service Addresses
- Service Distribution
- Measure performance, so you know how you are doing

Internet Mail

Six-Slide Mail Tutorial

Tutorial 1

- Mail User Agent (MUA)
  - what a user uses
  - sends mail using SMTP
  - receives mail using IMAP, or POP, or a file system

Tutorial II

- Mail Transfer Agent (MTA)
  - general term
  - receives mail using SMTP
  - sends mail to other hosts using SMTP
  - passes mail to a Mail Delivery Agent

Tutorial III

- Mail Delivery Agent (MDA)
  - accepts mail from an MTA
  - delivers it into some suitable database, ready for retrieval by other servers (IMAP, POP, web mail)

Tutorial IV

- SMTP is the Simple Mail Transfer Protocol
- simple, line-based textual protocol
  - HELO, MAIL FROM, RCPT TO, DATA
- Message Headers, Message Bodies
- Envelope Addresses

Tutorial V

- Appropriate servers for delivering mail are identified using the DNS, in general
  - MX records, A records
- MUAs usually use external MTAs to do this work for them, because they are stupid (”smart host”)
  - somewhat like stub resolver/caching resolver

Tutorial VI

- Spam makes people angry

Service Separation

- SMTP Servers can provide multiple functions, and in this tutorial we will separate them into four classes:
  - Relay Agent (”Smart Host”)
  - Mail Router (for outbound mail)
  - Mail Exchanger (for inbound mail)
  - Mail Store
    - with some appropriate delivery agent

Relay Agent

Function

- Accept mail from MUAs using SMTP
  - possibly using authentication (why?) or TLS (why?) or both (why?)
- Send that mail to a Mail Router
- Mess with users' messages
  - strip known-bad Outbreak Express viruses, raise red flags on spam
  - add irritating corporate messages

Names, Addresses

- Service referred to by clients using a DNS name
- The DNS name for the Relay Agent will typically be hard-coded, and it will be annoying for users to have to change it
- Renaming will probably be painful
    - we will need to support the old name for ever

Clients

- A well-known set of clients
  - “dial-up users”
  - “DHCP clients”
  - “workstations within our network”
  - “clients on the Internet who authenticate in some appropriate fashion”

Dependencies

- What must work in order for the Relay Agent to work?
  - Network access to clients
  - (access to some authentication service)
  - Network access to Mail Routers
  - Access to the DNS (Caching Resolvers)
- As long as clients and send mail, there is no immediate fault to complain about (maybe eliminate the DNS dependency for that)

Dependants

- What depends on the Mail Relay service?
  - Users being able to submit mail for sending
  - That's about it (good)

Availability

- The list of dependants is nice and small, so the availability of the Mail Relay service can be managed simply
- We can probably get a maintenance window

Growth

- As people send more mail, we will need to scale up our server
- As people send more ridiculous attachments, we will need to buy more disk
- For most enterprises this growth won’t require a very rapid increase in CPU or network
- For ISPs, the growth might be higher

Security

- We need to make sure we are careful who we relay for
- We need to make sure we are careful who we relay for
- We need to make sure we are careful who we relay for
- Maybe we'll use some port other than 25 (why?)

Data Gathering Over

- What are the implications for designing our service architecture?

Good Availability

- We need to be fairly sure we are up when we are supposed to be up
- Downtime for maintenance is acceptable, so long as we tell people (for enterprises)
- Downtime for maintenance may not be desirable, since people are stupid and don’t listen (for ISPs)

Name Stability

- Clients are going to hard-code the name of our Mail Relay in their MUAs
- There are lots of MUAs, and we really don't want to have to support them all over the phone
- Choose a simple name that is easy to spell, and try not to change it
  - some ISPs omit the domain part by default, and only give it to customers if the customers' operating system can't learn it

Service Addresses

- Not so dramatically useful for the Mail Relay service, since people use names rather than addresses, but still somewhat good
- We can change addresses in the DNS
  - but this relies on DNS stub resolvers and downstream caching resolvers being sensible
  - there is invariably a delay involved in DNS change propagation

Service Distribution

- We might decide to distribute Mail Relay servers to different places, so that we're not so dependent on the network performing well
  - SSL re-keying, TCP timeouts over high-latency links
- We might decide to deploy clusters of Mail Relay servers, so we don't need planned maintenance windows

Growth

- Add more CPU
- Add more disk
- Add more servers
- Add more (low-loss) network

Measurement

- Monitor queue lengths, free disk space, server load
- Test infrastructure
  - Attempt to relay mail every N minutes through the box to some collector box(es) which can sound alarms if N*M minutes pass without mail being received
  - Measure propagation delay (to off-net boxes, or via off-net auto-responders)
  - more than just ping

Summary

- Do not be an Open Relay
- Choose names wisely
- Use Service Addresses (maybe)
- Service Distribution (maybe)
- Measure performance, so you know how you are doing

Mail Router

Function

- Accept mail from Mail Relays
- Relay that mail to other SMTP servers in the Internet (or to local Mail Exchangers)
- Never touch message content
- Dedicated Outbound Mail Facility (why?)

Names, Addresses

- Service referred to by clients using a DNS name
- Renaming is trivial
- Names are not important
  - but don't make them too silly, since they will show up in Received headers

Clients

- A well-known set of clients
  - “Mail Relays”
  - that’s it

Dependencies

- What must work in order for the Mail Router to work?
  - Network access to clients
  - Network access to our Mail Exchangers (for local mail)
  - Network access to other peoples' Mail Exchangers (i.e. the Internet)
  - Access to the DNS (Caching Resolvers)

Dependants

- What depends on the Mail Relay service?
  - Mail Relays being able to send mail to other places
  - That's about it (good)

Availability

- The list of dependants is nice and small, so the availability of the Mail Router service can be managed simply
- We can definitely get a maintenance window
- If we're down for long, though, we put pressure on other servers' mail queues, and it might take time for our queues to clear
- Mail delays will eventually make the helpdesk phone ring (bad)

Growth

- Same as the Mail Relay
  - As people send more mail, we will need to scale up our server
  - As people send more ridiculous attachments, we will need to buy more disk
  - For most enterprises this growth won’t require a very rapid increase in CPU or network
  - For ISPs, the growth might be higher

Security

- We need to make sure we are careful who we relay for
- We should not relay for any hosts other than Mail Relays
- In fact, we should refuse SMTP connections from any SMTP client which we don't know
  - firewall the box off from the Internet

Data Gathering Over

- What are the implications for designing our service architecture?

Fair Availability

- We need to be fairly sure we are up when we are supposed to be up
- Downtime for maintenance is acceptable, so long as we don’t cause ourselves queue pain by being down for too long
- We don’t really need to notify anybody for short maintenance operations

Name Stability

- Largely irrelevant, however
  - Amusing and rude hostnames will appear in Received headers, so exercise some restraint
  - Difficult-to-spell or confusingly-similar hostnames will cause support headaches for your abuse department

Service Addresses

- Not spectacularly useful for the Mail Router service
- We can change addresses in the DNS
- However, if everything else is running on Service Addresses, maybe it’s easier not to be different

Service Distribution

- In order to avoid deep queues on Mail Relays, we probably want to deploy a Mail Router everywhere we have a Mail Relay
  - that way we have a nice place to worry about disk and CPU upgrades that the users don’t connect to
- We can deploy servers in clusters

Look Legitimate

- Make sure your server doesn’t look like a spam exploder
    - proper reverse DNS
    - handle abuse reports promptly and properly
- Use addresses which are not in similar ranges to those given out to customers

Growth

- Add more CPU
- Add more disk
- Add more servers
- Add more (low-loss) network
- Tune TCP stacks
- Use appropriate software, optimised to run queues efficiently

Measurement

- Monitor queue lengths, free disk space, server load
- Test infrastructure
  - Attempt to relay mail every N minutes through the box to some collector box(es) which can sound alarms if N*M minutes pass without mail being received
  - more than just ping

Summary

- Definitely do not be an Open Relay
- Use Service Addresses (maybe)
- Distribution of servers in coordination with Mail Relay servers
- Appear well-set-up (reverse DNS, not on blacklists)
- Measure performance, so you know how you are doing

Mail Exchanger

Function

- Accept mail from other servers on the Internet using SMTP
  - almost certainly without authentication or TLS (why?)
- Mess with users’ messages
  - strip known-bad Outbreak Express viruses, raise red flags on spam
- Deliver good mail to the Mail Store
- Dedicated Inbound Mail Facility (why?)

Names, Addresses

- Service referred to by clients using a DNS name
- Renaming will probably not be particularly painful (small DNS change)
  - we will need to support the old name a while (RR TTL*N)
  - sometimes renaming can be useful (spammers don't always use the DNS)

Clients

- A set of clients which cannot be enumerated
  - “other hosts on the Internet”
  - some are nice
  - some are evil (which are which?)

Dependencies

- What must work in order for the Mail Exchanger to work?
  - Network access to clients (i.e. the Internet)
  - Knowledge of what constitutes a local mailbox (cf dictionary attacks)
  - Access to the Mail Store service
  - Access to the DNS (Caching Resolver Service) (why?)

Dependants

- What depends on the Mail Exchange service?
  - Users being able to receive mail that was sent from other places
  - That’s about it (good)

Availability

- The list of dependants is nice and small, so the availability of the Mail Exchange service can be managed simply
- We can probably do brief maintenance without telling people, much
- Long maintenance periods willl result in mail queueing up in other places, and may cause noticable delays (helpdesk phone ring bad)

Growth

- As people receive more spam, uh, mail, we will need to scale up our server
- As people send our users more ridiculous attachments, we will need to buy more disk
  - not that much disk
  - we can use other peoples’ disk for this, to a large extent

Security

- We don’t relay for anybody
- Not anybody
- Not even ourselves

Spam

- The Mail Exchanger is the entry point for spam into our network
- (our Mail Relays can receive spam, but only from our own customers, and we know where they live)

Data Gathering Over

- What are the implications for designing our service architecture?

Good Availability

- We need to be fairly sure we are up when we are supposed to be up
- Downtime for maintenance is acceptable, and for short maintenance we might not need notification
- Longer periods of downtime are problematic, since users will notice
  - (helpdesk phone, etc, etc)

Name Stability

- Name stability is not particularly important
- Choice of name is not particularly important, except that, again,
  - people will read names from Received headers over the phone

Service Addresses

- Not spectacularly useful for the Mail Exchanger service
- We can change addresses in the DNS
- However, again, if everything else is running on Service Addresses, maybe it’s easier not to be different

Service Distribution

- If there are periods of unreachability for Mail Exchangers, then mail will get delayed
  - helpdesk phone rin, blah, blah
- The cost of queueing the mail that can’t be delivered is shared amongst all kinds of other people you don’t know
  - maybe you don’t care so much about that
- We can deploy servers in clusters
- We can add more MX records (probably)

Outbound Mail

- Mail Exchangers need to send mail (why?)
- Send the mail via Mail Routers, rather than queueing it yourself
  - concentrate all your queue handling on dedicated boxes

Growth

- Add more CPU (SpamAssassin eats CPU)
- Add more disk
- Add more servers to clusters
- Add more MXes (difficult?)
- Add more (low-loss) network

Measurement

- Monitor queue lengths, free disk space, server load
- Test infrastructure
  - Attempt to relay mail every N minutes through the box to some collector box(es) which can sound alarms if N*M minutes pass without mail being received
  - Attempt to relay mail to other places, and sound LOUD ALARMS when it works
  - more than just ping

Spam

- This is not a Spam Tutorial
- There will be no shouting, please

Spam

- There are some hosts we can refuse connections from, based on some criteria which makes sense for us
  - (reverse DNS, blacklists, etc)
- Other messages might require their message bodies to be examined before we can tell that they are spam
  - (DCC, SpamAssassin, Habeas, etc)
  - more expensive

Wildcard Mailboxes

- Some customers like to buy domains and arrange for mail addressed to *@their-domain to be delivered into one mailbox
  - very, very expensive to support
  - you will run out of disk very quickly
  - perl processes will consume all your CPU
  - you will become obsessed and constantly enraged by spammers, and you will never sleep again

Summary

- Do not relay at all, for anybody, ever
- Choose names wisely
- Use Service Addresses (maybe)
- Service Distribution (maybe)
- Measure performance, so you know how you are doing
- No wildcard mailboxes, if possible

Mail Store

Function

- Accept mail from Mail Exchangers using SMTP
- Deliver that mail to local mailboxes
- Make local mailboxes available to clients (IMAP, POP, web mail, etc)

Names, Addresses

- Service referred to by clients using a DNS name
- The name of the servers are really not important (this is for SMTP delivery, not IMAP/POP access)
  - usual comments on stupid names and Received headers apply

Clients

- A distinct, small set of clients who never call the helpdesk
  - Mail Exchangers
- We may well share hardware with IMAP/POP services
  - clients of those services do call the helpdesk

Dependencies

- What must work in order for the Mail Store to work?
  - Network access to clients (i.e. Mail Exchangers)
  - Knowledge of what constitutes a local mailbox in order to be able to deliver mail appropriately

Dependants

- What depends on the Mail Exchange service?
  - Users being able to receive mail that was sent from other places
  - That’s about it (good)

Availability

- The list of dependants is nice and small, so the availability of the Mail Store service can be managed simply
- We can probably get a maintenance window
- Customers will notice when the platforms go down (although not so much if just the Mail Store service goes down)

Growth

- As people receive more spam, uh, mail, we will need to scale up our server
- As people send our users more ridiculous attachments, we will need to buy more disk
  - lots and lots of disk
  - a strategy for removing unread mail from mailboxes after some time is probably sensible
  - strategy for refusing more mail to mailboxes which are already too full

Security

- We don’t relay for anybody
- Not anybody
- Not even ourselves
- We don’t even accept SMTP connections from anybody other than Mail Exchangers
- if we are liberal with this, users will start using the POP server to send mail through, and our dependencies just blew out

Data Gathering Over

- What are the implications for designing our service architecture?

Good Availability

- We need to be fairly sure we are up when we are supposed to be up
- The servers that the Mail Store service runs on will probably be customer-visible (POP, IMAP) so they have corresponding uptime requirements
- The Mail Store service itself can go down for short periods without people noticing

Name Stability

- Name stability is not particularly important
- Choice of name is not particularly important, except that, again,
  - people will read names from Received headers over the phone

Service Addresses

- Not spectacularly useful for the Mail Store service, but much more useful for IMAP/POP services
- We can change addresses in the DNS
- If everything else is running on Service Addresses, maybe it's easier not to be different

Service Distribution

- Distributing Mail Stores leads the slight problem that storing mail requires disk, and clients tend to like their mail to be stored in one place
  - Can distribute users amongst mail stores
    - support headache
  - Can distribute storage problem to dedicated boxes
    - network storage headache

IMAP/POP Proxy

- Use a proxy service to provide a consistent entry-point for users to retrieve mail
- Proxy servers can be distributed (clusters, anycast)
- Proxy servers connect to one of many POP/IMAP servers, depending on where the user's mail is kept
- Can move mailboxes without users knowing

Growth

- Add more CPU (maybe)
- Add more disk (definitely)
- Maybe distribute users between Mail Store services
  - POP/IMAP Proxies
  - Directory service for identifying location of user’s mail store (and existence of user, coincidentally, for Mail Exchanger)

Measurement

- Monitor queue lengths, free disk space, server load
- Test infrastructure
  - Attempt to relay mail every N minutes through the box to some collector box(es) which can sound alarms if N*M minutes pass without mail being received
  - Attempt to relay mail to other places, and sound LOUD ALARMS when it works
  - more than just ping

Spam

- Ideally we don’t worry about spam on the Mail Store at all -- it has already been dealt with by the Mail Exchanger
  - this means our IMAP/POP servers can be nice and snappy, unencumbered by the SpamAssassin CPU drain effect

Summary

- Do not relay at all, for anybody, ever
- Choose names wisely
- Use Service Addresses (maybe)
- Service Distribution (maybe, with proxies)
- Measure performance, so you know how you are doing

Summary of Summaries

General Approach

- Try to isolate big, complicated services into their component bits
  - not all components are visible by users
    - helpdesk phone ring bad
  - distribute performance management problem
  - distribute software selection problem
    - you can even run Exchange, if you want

DNS

- The most important service to get right
- Every service you provide to users, pretty much, depends on DNS
- When the DNS breaks, users notice
  - helpdesk phone ring bad

Internet Mail

- Arguably the most important application on the Internet
- Many of the components are very forgiving of transient network or server problems
- We need to be careful about what we send (we must punish our own spammers)
- We need to be careful about what we receive (we need to expect other people to be too lenient with their own spammers)

What Else?

- We haven’t talked about specific packages or products
  - that’s because the architecture is more important than the details of which package you use
    - separate functionality
    - use software that makes sense

Some Names

- BIND, other DNS implementations
- postfix, sendmail (semi-eek!), qmail, exim, Exchange (eek!), others
- mailscanner, mimedefang, DCC, SpamAssassin (well, ok we mentioned SpamAssassion), lots of others
- MAPS RBL and friends

The End

`http://www.isc.org/misc/netsa2003/dns-and-mail.pdf`

Joe Abley <jabley@isc.org>

INTERNET SOFTWARE CONSORTIUM