IntServ– General Framework

- Establish a contract for service (to tell routers what the application needs)
  - Guaranteed, predictive or best effort
- Admission control (decide if you can supply the service)
- Classify traffic (to tell what contract and buffer it belongs to)
- Schedule traffic (WFO ...)
- Policing, billing, ...

Describing the flow: Leaky Bucket Model

- Bucket leaks at a rate of $r$ packets per unit time
- Bucket can hold $B$ packets
- If packets arrive in a way that overfills the bucket, they are dropped
- Apply it to the flow examples above
- $r$ describes the average; $B$ the variability

QoS tools and prerequisites

- Metrics for Q in QoS: delay, bandwidth, but have to account for variability
- Tools for controlling those:
  - Queueing discipline (WRR, WFO)
  - Allocating resources (buffer space and bandwidth)
  - Require router support (data or control plane?)
Important Theoretical Result [Parekh92]

- If you know the nature of the traffic source (e.g., leaky bucket)
- By implementing WFQ scheduling, you can get an absolute upper bound on the network delay of the traffic in question
- Result generalizes to a network of WFQ routers!
- So, with WFQ we can get bandwidth and upper limit on delay guarantees!

Admission Control

- Can a new flow be granted requested QoS without impacting earlier guarantees?
- Does resource-reservation request satisfy local administrative policies?
- Is requestor authorized to make reservation?
- Implementing reservation
  - Assuming the worst case use of all currently admitted flows
  - Proposal: assuming the average use of admitted flows
  - Proposal: measure the actual use of the flows and make decisions based on that

Differentiated Services

- Quality of service associated with aggregates instead of a single flow
- Routers can then deal with these aggregates differently according to their service requirements
- Example: IP's expedited forwarding option (EF packets placed in a separate queue)
- Can use WFQ or simple priority to schedule them
- Police to some maximum peak rate, and drop offending packets

Differentiated Services

- Reference Implementation (RFC 1633)
  - Classifier
  - Packet Scheduler
  - Admission Control
  - Reservation Setup Protocol (RSVP)
This paper: Explore Tradeoffs for this type of solution

- Design decisions (which we won't cover)
  - Packet drops or packet marking to indicate congestion
  - Probe packet priority (lower or same as regular packets)
  - Probing Algorithm

- Also study the effect of router queueing mechanism

- Assumes a simple drop model where packets rate $r$ higher than capacity $C$ results in $r - C$ packet drops

Scalability/Deployment

- IntServ not scalable, but desirable—so, what has been done/can be done?

- Traffic Shapers/single point solutions (e.g., at ingress routers)

- Differentiated services + aggregation

- Emulating IntServ without requiring IntServ (e.g., Core Stateless Fair Queueing)
Is it worth Doing?

- Good: IntServ!!
- Good: Minimal router support
- Bad: Long setup time
- Bad: QoS not predictable in terms of delay (FIFO in the class)
- Ugly: enforcing that everyone plays nice
- Ugly: Deployment Barrier – does require changes to routers
  - to isolate from non-conforming flows
  - to keep out of TCP’s way (AC traffic is not TCP-friendly)

Router Queueing Discipline

- Consider FIFO and Fair Queueing – which is better?
- Two types of flows with rates $r_2 > r_1$, and $n_1, n_2$ active flows of each type
- FIFO: No flow is admitted if load $(n_1 \cdot r_1 + n_2 \cdot r_2 > C)$. No flow is admitted if it leads to lower performance to any flow
- FQ: susceptible to stolen bandwidth
  - An admitted flow of type 2 may get degraded performance as flows of type $r_1$ are admitted reducing its share of the bandwidth
- Prefer FIFO for end point admission control

Co-exsiting with Best-Effort Traffic

- Bandwidth will be stolen if best-effort traffic increases
- The sky is falling again? Incentives to play nice similar to the last paper we read?
- Aside: Should admission controlled flows be TCP friendly?
- Solution: Isolate best effort (BE) from admission controlled (AC) traffic
  - Should never allow AC to borrow bandwidth from BE because it will fool probes into letting in additional flows
  - Need something similar to fair queueing (Fair queueing with rate limit)
Key Insights

1. If each packet of a flow with arrival rate $r$ is forwarded with probability $P = \min(1, \frac{r}{f})$
   the expected rate of flow's forwarded traffic $r$ is $r' = r \cdot P = r \cdot \min(1, \frac{r}{f}) = \min(r, f)$
   Note that $f$ is the fair share for each flow.
2. No need to maintain per flow state at every node to estimate $r$, if $r$ is carried by the packet itself.
3. To maintain consistency of the estimated rate $r$, it is enough to updated it with $r'$ as the packet is forwarded.

Example

- Assume estimated fair rate $\tilde{f} = 4$
  - flow 1, $P = \min(1, 4/8) = 0.5$
  - expected rate of forwarded traffic $8 \cdot P = 4$
  - flow 2, $P = \min(1, 4/6) = 0.67$
  - expected rate of forwarded traffic $6 \cdot P = 4$
  - flow 3, $P = \min(1, 4/2) = 1$
  - expected rate of forwarded traffic $2$

Eliminating Per Flow Queuing
**Algorithm Details**

- Very simplified description
- $f$ has a monotonic impact on aggregate forwarded bandwidth $R$
- As $f$ increases, $R$ increases up to the link capacity
- Adapt $f$ based on congestion/no congestion determination
- How do Edges estimate $r$? exponential average

**Comparison to RED, FRED, FQ**

![Comparison Graphs]

**Domain Name System (DNS)**

- Scalable directory services protocol for the Internet
- Most common use: directory service to map host name to IP address
- Berkeley Internet Name Domain (bind) on Unix machines
- Domain Name System (DNS)

**Performance Evaluation**

![Network Diagram]
Challenges and Concerns

- **Challenges**
  - How to build a directory system for the whole internet?
  - Can you suggest some approaches (dumb or otherwise)?
    - HOSTS.txt – until mid-1980s
  - Is there a phone directory for the whole world?

- **Concerns/Requirements**
  - Ease of administration
  - Availability
  - Scalability
  - Security
  - Extensibility

---

Domain Name System Hierarchy

- Idea: use a hierarchy of domains; hosts defined within a domain
- Break the domains into zones
  - Each zone administered independently
- Each zone supported by two or more name servers (why?)
  - Primary and secondary are exact copies of each other
  - Secondary polls primary for updates

---

The naming system for the Internet

- Highly successful
- Widely distributed administration
- Good for long-lived, static information
- Not extensible
- Simple API

Name Servers and Resolvers

- Name servers are the directory databases
- Resolvers generate the queries that do the lookup (tree walk)

---

Each server maintains a collection of resource records (RR)

- Each record: (Name, Value, Type, Class, TTL)
  - Record indicates binding Name to Value
  - Type specifies the type of binding
  - Class allows other entities to define types
  - TTL: how long the record is valid for
**Resource Record Types**

- **A:** Value gives the 32-bit IPv4 address
- **PTR:** value gives hostname for the IP address in the name field
- **NS:** Value is the name for the host running the name server that knows how to resolve names within the specified domain name
- **CNAME:** provides canonical name for specified host; used for aliases
- **MX:** value gives the name for the host running mail server that accepts messages for the specified domain
- Not easily extensible; everyone must agree on changes

**MX Example**

- When you send mail to nael@opal.cs.binghamton.edu
- Mail program queries DNS for an MX record for opal
- The following info is returned (I used nslookup, querytype=mx):

  - **opal.cs.binghamton.edu**
    - canonical name = cs.binghamton.edu
  - **cs.binghamton.edu**
    - preference = 0, mail exchanger = cs.bing...
  - **cs.binghamton.edu**
    - internet address = 128.226.123.101
  - **cs.binghamton.edu**
    - name server=bingnet1.cc.binghamton.edu

- These correspond to:
  - (opal.cs.binghamton.edu.cs.binghamton.edu,CNAME,IN)
  - (cs.binghamton.edu.cs.binghamton.edu,MX,IN)
  - (cs.binghamton.edu.128.226.123.101,A,IN)
  - (cs.binghamton.edu.bingnet1...,NS,IN)

**Server Hierarchy**

- At the top level is the root domain managed by a set of root name servers
  - Give a starting point to the full DNS database
  - Thirteen servers distributed all over the world; why not just two?
  - Manages top level domains such as .edu, .com, .net and .org
  - Also manages geographical (country) domains such as .us, .in, and even .jo
- US domains are maintained by the Network Information Center (InterNIC)
- Next level is middle-level domains like binghamton.edu and yahoo.com
- Where do the root name servers get their information?
Who are you – reverse DNS

- the domain in-addr.arpa provides reverse mapping
- Used by servers to figure out who is connecting to them
- Records of type PTR
- nslookup with querytype=ptr on 128.226.123.101

| 101.123.226.128.in-addr.arpa | name = cs.binghamton.edu |
| 123.226.128.in-addr.arpa  | nameserver = bingnet1.cc.binghamton... |

- Dynamic DNS

Aside: DNS Cache Poisoning

- Spoofing attack (security next time)
- Mallory (man-in-the-middle) asks its DNS server for the address of hotmail.com
- DNS server recursively sends the request to dns.hotmail.com
- Before it can answer, Mallory spoofs the answer telling the DNS server that hotmail.com is his own machine
- Users now connect to Mallory instead of hotmail.com and volunteer their passwords
- Solution: authenticate DNS replies/updates

Name Resolution

- Site-wide cache to speed up resolution of frequently used names
- gethostbyname() and gethostbyaddr()

Caching and Replication

- 13 root servers (top level servers)
  - Recently, there was a denial of service attack on those
  - What would happen if the attack was successful?
- Caching: to reduce DNS traffic, each resolution is cached locally
  - Recommended TTL for hosts is 2 days
  - Record specifies TTL field (can set it to 0 if no caching is desired)
Why not DNS

- Presumes structure: permanent nodes in a hierarchy, root servers, etc...
- Requires manual management of routing information (NS records)
- Requires naming structure (nodes in .edu zones have to end with .edu)
- Updates difficult, dynamic DNS here but optional and not everywhere
- For P2P:
  - Hosts can come and go
  - No host hierarchy
  - No naming structure

Getting back to Naming

- DNS provides FQDN to IP name resolution; is this the only type of naming of interest?
- Service Discovery/Intentional Naming: equivalent to yellow pages
  - Find a service – typically not individualized
  - e.g., a printer, or a webserver ...
  - How would you implement it?
  - Would DNS work?

Some Peer to Peer Applications

- Cooperative Mirroring: content providers cache each other’s data – provide load balancing (plan for average rather than peak)
- Time-shared storage: if intermittently connected, someone else can make your “stuff” available. When you are connected, you make theirs available in return
- Distributed index: finding data based on keyword input as in Napster/Gnutella
- Embarrassingly parallel applications (e.g., breaking code, SETI@home)
Aside: SETI@home

- Largest and best known peer-to-peer computing project
- SETI: Search for Extraterrestrial Intelligence
  - Gobs of data from radio telescopes need to be analyzed
  - No one will fund them, can't buy computing resources
- Big success:
  - Around 4 mil. clients downloaded
  - Total CPU time: 1246848 years! 54.8 TFLOPS
  - ASCI RED (biggest supercomputing cluster) is 12 TFLOPS with cost $100 million
  - Get your own copy: http://setiathome.ssl.berkeley.edu/

Classifying Peer to Peer Networks

- Centralized vs. Decentralized
  - Centralized support or no centralization
  - For indexing and joining leaving

- Structured vs. Unstructured
  - Structured: data can be stored at a specific place (e.g., according to a hash function)
  - Unstructured: data has to be stored at a specific place

- Terms of Data and Network Structure
  - What is the tradeoff?

Searching in Decentralized Unstructured P2P Networks

- Example: (the old) Gnutella
  - Basically, we have no idea where the data is;
  - Devolves to random search

- Some approaches (assume existence of an overlay network connecting nodes)
  - Flood
  - Expanding Ring
  - Random Walkers
  - Local Indices
  - Replication of popular search items

- What if centralized unstructured?
  - Keep score of replies; you receive from neighbors; use highest scores on similar searches

Searching in Peer-2-Peer Networks

- What possible ways are there to organize a P2P network?
  - Napster – centralized index
  - Gnutella – flood the “overlay network”, get responses

- Neither is scalable, especially given the resolution and update frequency

- How do we carry out such generalized resolution in such a dynamic network?
Structured P2P

- Enforce a structure on the network, Typically using Distributed Hash Tables (DHT)

A DHT is a hash table where each node is responsible for a portion of the hash space
- A data item is placed at the node responsible for the hash value for its search key
  * Search key is predetermined – cannot do general searches
  * Finding a data item is quick – we know where it's stored!
    * But how do we know the responsible node?
    * What happens as nodes join and leave?

Main Improvement – Use Routing Pointers

- Use consistent hashing to map the key to a node
  * Each node is responsible for the keys that hash to the range (predecessor, id)
- Keep track of your power-of-2 successors in a routing table
  * log N entries per node; O(Log N) messages needed to resolve
- Joins/Stabilization in the background to keep pointers current

Chord Protocol

- Key idea:
  - Arrange the nodes in a logical circle, where every node knows its successor node
    * Pass the request along the circle until someone replies
    * Very slow: O(N) messages needed, sequential resolution

Discussion

- Structured vs. unstructured
- Hash is random, neighbors may be very far from each other
- What about restricting where the data goes (e.g., for security reasons?)
Other Ideas and Open Problems

- Other Structured Organizations have been studied
  - DHT based: CAN (Hypercube), Tapestry (Plaxon Tree), Debruijn Network, XOR Network, ...
  - Skip Lists
  - SIGCOMM 2003 paper argues that ring is best from a robustness perspective

- Loosely structured (super-peers that are structured)?

- Incentives for participation?

- Freenet, BT...