Network Security
mod-net-sec

Lecture 6
Peer-to-peer networks-I (Gnutella)

Shishir Nagaraja
Jan 29, 2013
Acknowledgements

• The slides during this semester are based on ideas and material from the following sources:
  – Slides from Professor S. Gosh’s course at University of Iowa.
Administrivia

• Project work
  – Deadlines
  – Scoring guidelines / assessment criteria
Plan for today

• Failure detection in networked systems and why you need it
• P2P networks
  – Napster – the parent of all p2p networks
  – Gnutella – an unstructured p2p network
  – Chord
  – Attacks and defenses
Two Different System Models

- **Synchronous Distributed System**
  - Each message is received within bounded time
  - Each step in a process takes $lb < time < ub$
  - (Each local clock’s drift has a known bound)

- **Asynchronous Distributed System**
  - No bounds on process execution
  - No bounds on message transmission delays
  - (The drift of a clock is arbitrary)

*The Internet is an asynchronous distributed system*
Failure Model

- **Process omission failure**
  - **Crash-stop (fail-stop)** – a process halts and does not execute any further operations
  - **Crash-recovery** – a process halts, but then recovers (reboots) after a while

- *Crash-stop* failures can be detected in synchronous systems

- Next: detecting *crash-stop* failures in asynchronous systems
Network partition

Crashed router
What’s a failure detector?

$pi$

$ pj $
What’s a failure detector?

Crash-stop failure

\( \pi \)
What’s a failure detector?

needs to know about \(pj\)’s failure

Crash-stop failure
I. Ping-Ack Protocol

- *pi* queries *pj* once every *T* time units
- if *pj* does not respond within *T* time units, *pi* marks *pj* as failed

If *pj* fails, within *T* time units, *pi* will send it a ping message, and will time out within another *T* time units. Detection time = 2*T*
II. Heart-beating Protocol

- pj maintains a sequence number
- pj sends pi a heartbeat with incremented seq. number after every $T'(=T)$ time units

If pj has sent $x$ heartbeats until the time it fails, then pi will timeout within $(x+1)T$ time units in the worst case, and will detect pj as failed.
Failure Detector Properties

- **Completeness** = every process failure is eventually detected (no misses)
- **Accuracy** = every detected failure corresponds to a crashed process (no mistakes)
- Given a failure detector that satisfies both Completeness and Accuracy
  - One can show that Consensus is achievable
  - FLP => one cannot design a failure detector (for an asynchronous system) that guarantees both above properties
Completeness or Accuracy?

• Most failure detector implementations are willing to tolerate some inaccuracy, but require 100% completeness

• Plenty of distributed apps designed assuming 100% completeness, e.g., p2p systems
  – “Err on the side of caution”.
  – Other processes need to make repairs whenever a failure happens

• **Heart-beating** – satisfies completeness but not accuracy (why?)

• **Ping-Ack** – satisfies completeness but not accuracy (why?)
Completeness or Accuracy?

• Both Heart-beating and Ping-Ack provide
  – *Probabilistic accuracy* (for a process detected as failed, with some probability close to 1.0, it is true that it has actually crashed).
  – That was for asynchronous systems

• Heart-beating and Ping-ack can *satisfy both completeness and accuracy in synchronous systems* (why?)
Failure Detection in a Distributed System

• Difference from original failure detection is
  – we want not one process ($pi$), but all processes in system to know about failure

➤ ➞ May need to combine failure detection with a dissemination protocol
  – What’s an example of a dissemination protocol?
Failure Detection in a Distributed System

• Difference from original failure detection is
  – we want not one process ($pi$), but all processes in system to know about failure

➢ May need to combine failure detection with a dissemination protocol
  – What’s an example of a dissemination protocol?
    • A reliable multicast protocol!
Centralized Heart-beating

Needs a separate dissemination component
Downside?
Ring Heart-beating

$\text{pj, Heartbeat Seq. } l++$

$\text{pj}$

$\text{pi}$

$\text{Needs a separate dissemination component}$

$\text{Downside?}$
All-to-All Heart-beating

Does not need a separate dissemination component

Downside?
Efficiency of Failure Detector: Metrics

• **Measuring Speed**: Detection Time
  – Time between a process crash and its detection
  – Determines speed of failure detector

• **Measuring Accuracy**: depends on distributed application
Accuracy Metrics

- **Tmr**: Mistake recurrence time
  - Time between two consecutive mistakes

- **Tm**: Mistake duration time
  - Length of time for which correct process is marked as failed (for crash-recovery model)
More Accuracy Metrics

• **Number of false failure detections** per time unit (**false positives**)
  – System reported failure, but actually the process was up
  – Failure detector is inaccurate

• **Number of not detected failures** (**false negatives**)
  – System did not report failure, but the process failed
  – Failure detector is incomplete
Processes and Channels

process $p$

send $m$

Communication channel

process $q$

receive

Outgoing message buffer

Incoming message buffer
Other Failure Types

- **Send-omission**: loss of messages between the sending process and the outgoing message buffer (both inclusive)
  - What might cause this?

- **Channel omission**: loss of message in the communication channel.
  - What might cause this?

- **Receive-omission**: loss of messages between the incoming message buffer and the receiving process (both inclusive)
  - What might cause this?
Other Failure Types

- **Arbitrary Failures (Byzantine)**
  - Arbitrary process failure: arbitrarily omits intended processing steps or takes unintended processing steps.
  - Arbitrary channel failures: messages may be corrupted, duplicated, delivered out of order, incur extremely large delays; or non-existent messages may be delivered.

- Above two are Byzantine failures, e.g., due to hackers, man-in-the-middle attacks, viruses, worms, etc.

- A variety of Byzantine fault-tolerant protocols have been designed in literature!
  - “Scaling Byzantine Fault-tolerant replication in WAN”, DSN 2006
# Omission and Arbitrary Failures

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop or Crash-stop</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect this state.</td>
</tr>
<tr>
<td>Omission</td>
<td>Channel</td>
<td>A message inserted in an outgoing message buffer never arrives at the other end’s incoming message buffer.</td>
</tr>
<tr>
<td>Send-omission</td>
<td>Process</td>
<td>A process completes a <code>send</code>, but the message is not put in its outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission</td>
<td>Process</td>
<td>A message is put in a process’s incoming message buffer, but that process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary (Byzantine)</td>
<td>Process or channel</td>
<td>Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.</td>
</tr>
</tbody>
</table>
Timing Failures

• In synchronous distributed systems - applicable
  – Need **time limits** on process execution time, message delivery time, clock drift rate

• In asynchronous distributed systems - not applicable
  – Server may respond too slowly, but we cannot say if it is timing failure since no guarantee is offered

• In real-time OS - applicable
  – Need timing guarantees, hence may need redundant hardware

• In multimedia distributed systems – applicable
  – Timing important for multimedia computers with audio/video channels
# Timing Failures

<table>
<thead>
<tr>
<th>Class of Failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Process</td>
<td>Process’s local clock exceeds the bounds on its rate of drift from real time</td>
</tr>
<tr>
<td>Performance</td>
<td>Process</td>
<td>Process exceeds the bounds on the interval between two steps</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
<td>A message’s transmission takes longer than the stated bound</td>
</tr>
</tbody>
</table>
Masking Failures

• It is possible to construct **reliable services** from components that exhibit failures
  – Example: multiple servers that hold replicas of data can continue to provide a service when one of the crashes

• **Knowledge of failure characteristics** of a component can enable a new service when one of them crashes

• Service **masks** failure either by **hiding** it altogether or **converting** it into more acceptable type of failure
  – Example: checksums mask corrupted messages, effectively converting an arbitrary failure into omission failure

• **Reliability of one-to-one communication**
  – Defined in terms of validity and integrity
    • **Validity** – any message in the outgoing message buffer is eventually delivered to the incoming message buffer
    • **Integrity** – message received is identical to one sent, and no messages are delivered twice
Wrapping up failure detection

• **Failure detectors** are required in distributed systems to maintain liveness and transparency in spite of process crashes

• Properties – **completeness & accuracy**, together unachievable in asynchronous systems

• Most apps require **100% completeness**, but can tolerate inaccuracy

• 2 failure detector algorithms – Heart-beating and Ping-Ack

• Distributed Failure Distribution through heart-beating algorithms: **Centralized, Ring, All-to-all**

• Accuracy metrics

• Other Types of Failures
Peer-to-peer networks:
Napster and the Gnutella protocol
Two Angles of Distributed Systems

D.S. Theory

(Next few lectures)
Peer to peer systems
Some Questions

• Why do people get together?
  – to share information
  – to share and exchange resources they have
    • books, class notes, experiences, music cd’s

• How can computers help people
  – find information
  – find resources
  – exchange and share resources

• Need Search Capabilities in Distributed Systems!
Current State of the Art

• **Conventional D/S technologies:** The **Web**!
  – Search engines
  – Forums: chat rooms, blogs, ebay
  – Online business
  – **Well organized content**

• **Services provided by users**
  – **Disorganized content**
Peer-to-Peer Systems

• If you can find a peer who wouldn’t mind exchanging one of her music files for one of yours, that would be great!

• **Napster: a light weight solution (lighter than the Web)**
P2P Systems Classification

• Hybrid
  – Centralized index, but P2P file storage and transfer
  – eg. limewire, napster

• Pure
  – Functionality completely distributed
  – Tor, bittorrent

• Super-peer or Hierarchical
  – A “pure network of “hybrid” clusters
  – Skype, Storm botnet
Metrics for Search/Insertion/Join

• **Cost (aggregate)**
  – Number of messages or interactions
  – Bandwidth
  – Processing power

• **Quality of Results**
  – Number of results
  – Satisfaction (true if # results >= X, false otherwise)
  – Time to satisfaction
Napster Brief History

- [6/99] Shawn Fanning (freshman Northeastern U.) releases Napster online music service
- [12/99] RIAA (Recording Industry Association of America) sues Napster, asking $100K per download
- [3/00] 25% UWisc traffic Napster, many universities ban it
- [00] 60M users
- [2/01] US Federal Appeals Court: users violating copyright laws, Napster is abetting this
- [9/01] Napster decides to run paid service, pay % to songwriters and music companies
- [Today] Napster protocol is open, people free to develop opennap clients and servers http://opennap.sourceforge.net
- [Today] eDonkey, BitTorrent, …
**Napster Structure**

*Store a directory, i.e., filenames with peer pointers*

<table>
<thead>
<tr>
<th>Filename</th>
<th>Info about</th>
</tr>
</thead>
<tbody>
<tr>
<td>PennyLane.mp3</td>
<td>Beatles, @ 128.84.92.23:1006</td>
</tr>
</tbody>
</table>

- napster.com Servers (Index Servers)
- Client machines (“Peers”)

Store their own files
Napster Operations

• **Client**
  – Connect to a Napster server (with well-known public address)
  – Upload list of music files that you want to share (names only, not the files themselves!)

• **Server** maintains list of `<filename, ip_address, portnum>` tuples

• **Search Protocol** from a client:
  – Send server keywords to search with
  – (Server searches its directory with the keywords)
  – Server returns a list of matching hosts –
    • `<ip_address, portnum>` tuples to client
  – Client pings each host in the list to find transfer rates
  – Client fetches file from best host

• All communication uses TCP/IP
<table>
<thead>
<tr>
<th>Filename</th>
<th>Filesize</th>
<th>Bitrate</th>
<th>Freq</th>
<th>Length</th>
<th>User</th>
<th>Connection</th>
<th>Ping</th>
</tr>
</thead>
<tbody>
<tr>
<td>incomplete_other_artist\Tito Puentes Golden Latin Jazz Allstars - Oye Como ...</td>
<td>3,696,640</td>
<td>128</td>
<td>44100</td>
<td>3:51</td>
<td>bdenzler</td>
<td>DSL</td>
<td>343</td>
</tr>
<tr>
<td>incomplete_other_artist[Marty Robbins] The Fastest Gun Around.mp3</td>
<td>542,304</td>
<td>128</td>
<td>44100</td>
<td>0:39</td>
<td>bdenzler</td>
<td>DSL</td>
<td>343</td>
</tr>
<tr>
<td>incomplete_other_artist\Ravi Shankar - Chants Of India 04 - Asato Maa.mp3</td>
<td>2,449,408</td>
<td>128</td>
<td>44100</td>
<td>2:35</td>
<td>bdenzler</td>
<td>DSL</td>
<td>343</td>
</tr>
<tr>
<td>other artist\Engelbert Humperdinck - White Christmas.mp3</td>
<td>9,277,648</td>
<td>320</td>
<td>44100</td>
<td>3:52</td>
<td>bdenzler</td>
<td>DSL</td>
<td>343</td>
</tr>
<tr>
<td>other artist\Grateful Dead - Franklin's Tower - Reggae Style.mp3</td>
<td>4,635,458</td>
<td>128</td>
<td>44100</td>
<td>4:48</td>
<td>bdenzler</td>
<td>DSL</td>
<td>343</td>
</tr>
<tr>
<td>Unknown Artist - You seriously have to listen to this.mp3</td>
<td>462,848</td>
<td>318</td>
<td>16000</td>
<td>0:17</td>
<td>sam113...</td>
<td>Cable</td>
<td>383</td>
</tr>
<tr>
<td>MP3z\artist - 'The Way Life Is' By Drag-On featuring Case.mp3</td>
<td>4,726,784</td>
<td>128</td>
<td>44100</td>
<td>4:54</td>
<td>burg651</td>
<td>Cable</td>
<td>386</td>
</tr>
<tr>
<td>MP3z\artist - 'Opposite Of H20' By Drag-On featuring Jadakiss.mp3</td>
<td>3,540,992</td>
<td>128</td>
<td>44100</td>
<td>3:41</td>
<td>burg651</td>
<td>Cable</td>
<td>386</td>
</tr>
<tr>
<td>Various Artist - Perfect Day 97.mp3</td>
<td>3,722,344</td>
<td>128</td>
<td>44100</td>
<td>4:05</td>
<td>falkstad</td>
<td>ISDN-128K</td>
<td>398</td>
</tr>
<tr>
<td>Liszt\Liszt - Etude 'Un sospiro' - Cziffra-artist.mp3</td>
<td>2,752,512</td>
<td>128</td>
<td>44100</td>
<td>2:53</td>
<td>Iskdflikj...</td>
<td>Unknown</td>
<td>504</td>
</tr>
<tr>
<td>Music\Waiting To Exhale - Original Soundtrack Album - Various Artist - Count...</td>
<td>3,199,083</td>
<td>96</td>
<td>44100</td>
<td>4:26</td>
<td>Jzfork9</td>
<td>56K</td>
<td>511</td>
</tr>
<tr>
<td>Track_03_artist.mp3</td>
<td>4,054,332</td>
<td>128</td>
<td>44100</td>
<td>4:13</td>
<td>immusic...</td>
<td>Cable</td>
<td>514</td>
</tr>
<tr>
<td>Track_02_artist.mp3</td>
<td>6,228,974</td>
<td>128</td>
<td>44100</td>
<td>6:26</td>
<td>immusic...</td>
<td>Cable</td>
<td>514</td>
</tr>
<tr>
<td>Track_01_artist.mp3</td>
<td>4,731,426</td>
<td>128</td>
<td>44100</td>
<td>4:54</td>
<td>immusic...</td>
<td>Cable</td>
<td>514</td>
</tr>
<tr>
<td>Track_04_artist.mp3</td>
<td>4,514,505</td>
<td>128</td>
<td>44100</td>
<td>4:41</td>
<td>immusic...</td>
<td>Cable</td>
<td>514</td>
</tr>
<tr>
<td>Track_05_artist.mp3</td>
<td>4,105,323</td>
<td>128</td>
<td>44100</td>
<td>4:16</td>
<td>immusic...</td>
<td>Cable</td>
<td>514</td>
</tr>
<tr>
<td>mixer in track_01_Artist_0721011750.mp3</td>
<td>180,686</td>
<td>128</td>
<td>44100</td>
<td>0:17</td>
<td>immusic...</td>
<td>Cable</td>
<td>514</td>
</tr>
<tr>
<td>Album\Reflex - Keep In Touch-Artist.mp3</td>
<td>7,041,024</td>
<td>160</td>
<td>44100</td>
<td>5:43</td>
<td>rotimco</td>
<td>56K</td>
<td>527</td>
</tr>
</tbody>
</table>

Returned 100 results.
Napster Search

2. All servers search their lists (ternary tree algo.)
   - Store peer pointers for all files

napster.com Servers (Index Servers)

Peers

3. Response
1. Query

4. ping candidates
5. download from best host

Store their own files
Problems

• Centralized server – source of congestion
• Centralized server – prone to security failures
• Centralized server – legal weakness
• No security: plaintext messages and passwds
• napster.com responsible for abetting users’ copyright violation
  – “Indirect infringement”
Gnutella

• **Unstructured** Peer-to-Peer System (in terms of search capabilities) - File sharing network
• Eliminates the servers
• Client machines search and retrieve amongst themselves
• Clients act as servers too, called **servents**
• [3/00] release by AOL, immediately withdrawn, but 88K users by 3/03
• Original design underwent several modifications; we’ll look at the initial version

http://www.limewire.com
Gnutella

Servents ("Peers")

Connected in an overlay graph

Store their own files

Also store "peer pointers"

a peer pointer
How do I search for my Beatles file?

- Gnutella routes different messages within the overlay graph
- Gnutella protocol has 5 main message types
  - Query – search for a file
  - QueryHit - response to query
  - Ping - discover hosts on network; to probe network for other peers
  - Pong (reply to ping, contains address of another peer)
  - Push - download request for firewalled servents (used to initiate file transfer)
- We’ll go into the message structure and protocol now (note: all fields except IP address are in little-endian format)
Gnutella Message Header Format

Descriptor Header

- Descriptor ID
- Payload descriptor
- TTL
- Hops
- Payload length

Payload

- ID of this search transaction
- Type of payload
  - 0x00 Ping
  - 0x01 Pong
  - 0x40 Push
  - 0x80 Query
  - 0x81 Queryhit

- TTL
- Hops
- Payload length

**ttl(0) = ttl(i) + hops(i)**

- Decremented at each hop,
  - Message dropped when ttl=0
- ttl_initial usually 10
## Payload Format in Gnutella Query Message

### Example: Query (0x80)

<table>
<thead>
<tr>
<th>Minimum Speed</th>
<th>Search criteria (keywords)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Description Header

<table>
<thead>
<tr>
<th>DescID</th>
<th>Minimum Speed</th>
<th>Search criteria (keywords)</th>
<th>Payload (Query)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80</td>
<td>10</td>
<td>0</td>
<td>128kb/s</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>PennyLane.mp3</td>
<td></td>
</tr>
</tbody>
</table>
Gnutella Search

Query’s flooded out, ttl-restricted, forwarded only once

$TTL=1$ (Query)

Who has PennyLane.mp3?

Requestor (Starts Search)
Payload Format in Gnutella Query Reply Message

QueryHit (0x81) : successful result to a query

<table>
<thead>
<tr>
<th>Num. hits</th>
<th>port</th>
<th>ip_address</th>
<th>speed</th>
<th>(fileindex, filename, fsize)</th>
<th>servent_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>n+16</td>
</tr>
</tbody>
</table>

Results

Info about responder

Unique identifier of responder; a function of its IP address

Example:

<table>
<thead>
<tr>
<th>DescID</th>
<th>0x81</th>
<th>10</th>
<th>0</th>
<th>100</th>
<th>1</th>
<th>1033</th>
<th>208.17.50.4</th>
<th>10Mbps</th>
<th>…..</th>
</tr>
</thead>
</table>
Successful results QueryHit’s routed on reverse path.

Who has PennyLane.mp3?
Gnutella Search

Successful results QueryHit’s routed on reverse path

Who has PennyLane.mp3?
Avoiding Excessive traffic

• To avoid duplicate transmissions, each peer maintains a list of recently received messages.
• Query forwarded to all neighbors except peer from which received (and this is remembered).
• Each Query (identified by DescriptorID) forwarded only once.
• QueryHit routed back only to peer from which Query received with same DescriptorID.
• Duplicates with same DescriptorID and Payload descriptor (msg type) are dropped.
• QueryHit with DescriptorID for which Query not seen is dropped.
Dealing with Firewalls

Requestor sends **Push** to responder asking for file transfer

Has PennyLane.mp3

But behind firewall

(Why is the Push routed and not sent directly?)
Push Message Format

Push (0x40)

<table>
<thead>
<tr>
<th>servent_id</th>
<th>fileindex</th>
<th>ip_address</th>
<th>port</th>
</tr>
</thead>
</table>

same as in received QueryHit

Address at which requestor can accept incoming connections
Dealing with Firewalls

• Responder establishes a TCP connection at ip_address,port specified. Sends

        GIV  <File Index>:<Servent Identifier>/<File Name>

• Requestor then sends GET to responder (as before) and file is transferred

• What if requestor is behind firewall too?
  – Gnutella gives up
  – Can you think of an alternative solution?
Ping (0x00)

no payload

Pong (0x01)

<table>
<thead>
<tr>
<th>Port</th>
<th>ip_address</th>
<th>Num. files shared</th>
<th>Num. KB shared</th>
</tr>
</thead>
</table>

• Peers initiate Ping’s periodically

• Ping’s flooded out like Query’s, Pong’s routed along reverse path like QueryHit’s

• Pong replies used to update set of neighboring peers
Summary of Control Messages (Ping/Pong/Query/Hit Routing)

Example 1. Ping/Pong Routing

Example 2. Query/QueryHit/Push Routing
After receiving QueryHit messages

- **Requestor chooses best QueryHit responder**
  - Initiates HTTP request directly to responder’s ip+port (file data never transferred over Gnutella network)
    
    ```
    GET /get/<File Index>/<File Name>/HTTP/1.0
    Connection: Keep-Alive
    Range: bytes=0-filesize
    User-Agent: Gnutella
    
    HTTP 200 OK
    Server:Gnutella
    Content-type:application/binary
    Content-length: 1024
    
    HTTP is the file transfer protocol. Why?
    Why the “range” field in the GET request?
    What if responder is behind firewall that disallows incoming connections?
Gnutella Summary

- No index servers
- Peers/servents maintain “neighbors” (membership list), this forms an overlay graph
- Peers store their own files
- Queries flooded out, ttl restricted
- Query Replies reverse path routed
- Supports file transfer through firewalls (one-way)
- Periodic Ping-Pong to keep neighbor lists fresh in spite of peers joining, leaving and failing
  - List size specified by human user at peer: heterogeneity means some peers may have more neighbors
  - Gnutella found to follow power law distribution:

\[ P(\text{#neighboring links for a node} = L) \sim L^{-k} \quad (k \text{ constant}) \]
Summary

• Napster: protocol overview, more details available on webpage
• Gnutella protocol
• Protocols continually evolving, software for new clients and servers conforming to respective protocols: developer forums at
  – Napster: http://opennap.sourceforge.net
  – Gnutella: http://www.limewire.com
• Others
  – Peer to peer working groups: http://www.p2pwg.com