Peer-to-Peer Networking

P2P Issues

Lecture Content

- So far we have seen numerous different P2P applications and systems
- Time to move focus on the issues behind most of the P2P systems
  - Being able to *communicate*,
  - *name and find* everyone and everything, then peers
  - wish to *stay connected*, and
  - do all this in a *secured* way (next lecture).
  - Shared resources still need to be *cooperative managed*

Communication

- Communication between peers is the heart of P2P
- Why it's complex?
  - The growth of the Internet led to different addressing schemes: dynamic and private network addresses
  - Security needs introduced firewalls, NATs and proxies
- Led to heterogeneity of networks
  - communication within enterprise intranet, extranet and open Internet is different
- Direct exchange between peers gets harder

Addresses and Names

- Expected that you are already familiar with these
  - More detailed in, for example, Barkai's book
- TCP/IP is the *de facto* standard network protocol, on which most of P2P will rely
- Addresses on the Net
  - Limited address space
  - Assigning temporary, dynamic addresses and setting a private subnet to save addresses
  - DNS translates them into names

Protocols and Standards

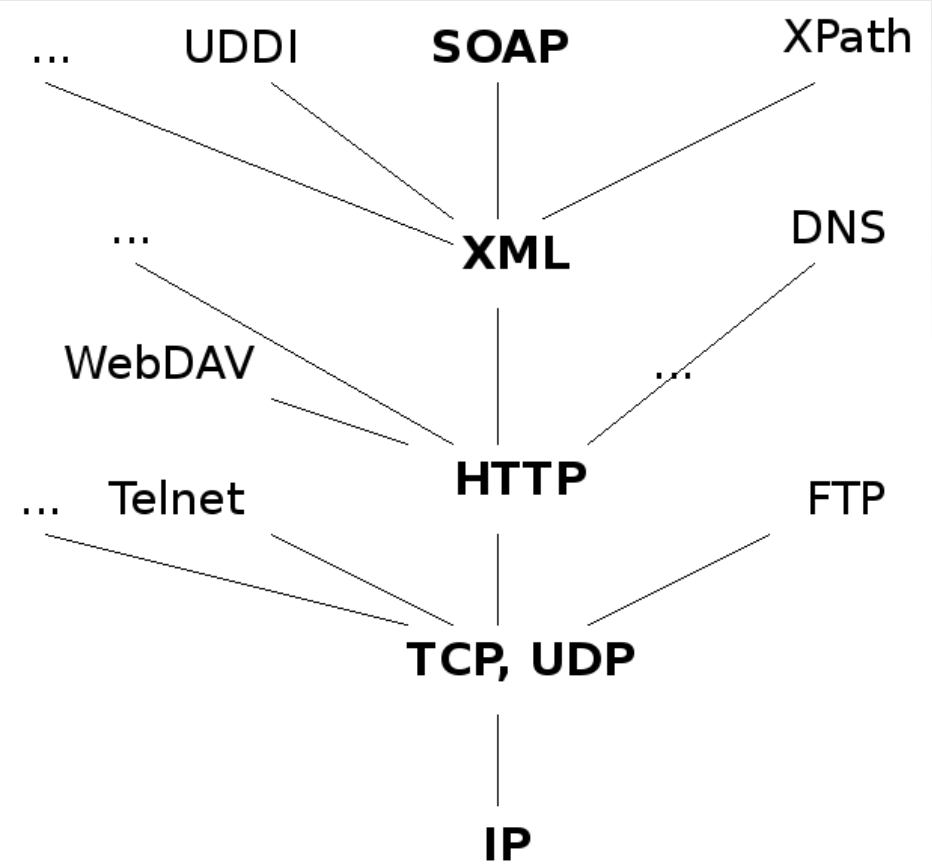
- Standard protocols simplify and accelerate development for multiple platforms
- A short reminder of popular application layer protocols used by P2P developers
- HTTP (Hyper Text Transfer Protocol)
  - Set of rules for exchanging files on the web
- XML (Extensible Markup Language)
  - Extensible markup language: describe the logical structure of a document

Protocols and Standards

- SOAP (Simple Object Access Protocol)
  - Builds on XML and HTTP offering light-weight message exchange in a decentralised and distributed setting
- WebDAV (Distributed Authoring and Versioning)
  - Extensions to HTTP: collaboratively manage files on Web servers (add *write* to HTTP's *read*)
- UDDI (Universal Description, Discovery and Integration)
  - For P2P: discovery of resources

The Protocol Tree

- A useful way to remember the roles and relationship of some of the protocols interesting to P2P
- TCP and UDP most common transtport protocols, sitting on top of IP
- Most convenient network protocol is probably HTTP
- P2P employs WebDAV, DNS and XML on top HTTP
- XML is a building block for many, like UDDI, SOAP and XPath



Firewalls and NATs

- NAT device allows multiple machines to communicate with the Internet using a single globally unique IP address
  - Solve the scarce IPv4 address space problem
  - Conceal internal network and resources
- Firewall prevents outsiders from accessing resources in a network, may also control the outbound access
- Both usually located at a gateway to the internal network as an integrated service

What Is Wrong with NAT?

- Breaks the Internet more than it makes it
- Can't access machines behind a NAT
  - NAT will let traffic reach the appropriate private IP only if there is mapping between the private IP/TCP (UDP) port number with the NAT's public IP/TCP (UDP) port number
  - Mapping created only when traffic originates from the private IP to the Internet - not vice versa
  - Designed to be transparent, not something to be selected
- Beyond access, a bunch of uncertainties lies ahead...

What Is Wrong with NAT?

- Symmetric NAT simply drops all unsolicited traffic coming from the Internet to the private address
  - Security through obscurity
  - Harder to determinate what goes wrong
- No standardized NAT behaviour, several flavours
  - Full cone NAT, restricted cone NAT, port restricted cone NAT, and symmetric NAT, NATs can be nested...
  - Access attempts through trial and error?
- NAT traversal is a bit easier with UDP than TCP

NAT and Firewall Traversal

- Allowing inbound traffic
  - Mimic a scenario in which the target host originates a connection to us and then we connect to it as though we are responding to the request
- Known as UDP hole punching
  - Not a security breach, but sensibly way to solve the issue
  - Get the private host to originate traffic so we can send our connection request to it masquerading as a reply
  - NATs also have an idle timer, after which reply is dropped

Using a Rendezvous Server

- Rendezvous (mediator) server listening on a globally routable IP address
- Private IPs can almost always open TCP connections to global IP addresses
- All the P2P nodes are constantly in touch with a rendezvous server, listening on a global IP address through a persistent TCP connection
- Through this TCP connection inform a particular P2P node that another node wants to talk to it

Hole Punching: One NAT

- Assume that we are not behind NAT, but want to communicate with a peer behind a NAT device
  - Inform the peer through the rendezvous server
  - Target node sends a request following which we send the connection request as a response to the request
  - In order to response back, determine the port assigned at the NAT's public interface for the private IP connection
    - By inspecting the source address of the UDP datagram that reaches any global IP
- NAT allows communication (remember idle timer)

Hole Punching: Two NATs

- But, commonly both peers are behind NAT?
- Send a packet to the rendezvous server (UDP)
  - Creates a global IP/port combo at the NAT public interface
- Mediator replies us back with our public IP/port
- Tell the mediator to request the P2P target node to initiate a request to us at that public IP/port
- Subsequently, we can connect to it as a reply to that message?

Hole Punching: Two NATs

- Only a full cone NAT would allow that... most won't
- So far we have:
  - Both peers know others public IP/port
  - But initiating communication drops at the end-point NAT
- Solution: Both nodes sends packets to each other's public IP/port at the same time
  - The first packet from each party discarded as unsolicited
  - But subsequent packets are let through because NAT thinks the packets are replies to our original request

Notes for Hole Punching

- The above described the fundamental procedure, but there's still some work to do
- First, figure out how the NAT behaves
  - Send probe packets to two different IPs
  - If behaviour is consistent, we are good to go, otherwise:
- Bumped into symmetric NAT behaviour that varies the port between requests?
  - Figure out the delta by which the port number varies and guess the port assigned for a particular request

NAT traversal in General

- The described method is similar to STUN (Simple Traversal of UDP over NATs) protocol, RFC 3489
  - STUN is more complicated, trying to detect different NAT types
- Other solutions exist (left as an exercise :)
  - Traversal Using Relay NAT (TURN)
  - NAT traversal based on NAT control (Realm-Specific IP, Middlebox Communications, Universal Plug and Play)
- No working generic solution yet...

Summary: Communication

- P2P systems heavily dependent to communication and ubiquitous connectivity
- Heterogeneity of networks
  - Dynamic addressing, firewalls, NATs
  - Preventing true end-to-end communications
  - No general solutions, but may work on most of the time
    - UDP Hole Punching
- P2P protocols rely on the standard protocols
  - Convergence to a set of protocols, HTTP is central

Naming and Finding

- Naming on the Internet is based on DNS
  - Server organised in a hierarchical manner, ISPs cache information locally
  - DNS works fine with fixed, permanent IP addresses
  - Suitable for client-server as long as server is fixed
- Breaks down when it comes to P2P systems
  - Dynamic addresses propagate slowly across DNS servers, peers cannot find others
  - DNS cannot deal with intermittent connectivity

Naming Schemes for P2P

- Instant Messaging
  - Peers registered with a unique name (e.g. email address) to that application environment
  - When user connects, name is mapped to the current IP address
- DNS names machines... See the difference?
  - P2P needs separating the user identify from the physical location of a host device
  - Able to follow a resource as it moves around

Naming Schemes for P2P

- Important to replace DNS with decentralised naming mechanisms, that are able to name all kind of objects
  - Supports computing on the edge of the network
- P2P naming based on rules, or protocol
  - Mapping to physical location takes place when an IP address is associated with the name of the object or a user
- Current solutions are application specific and define their own name spaces

DHT-based DNS

- Chord-based DNS
  - Lookup service, with host names as keys and IP addresses (and other host information) as values
  - Can efficiently fetch a matching value for a key
  - Require no special servers, while ordinary DNS systems rely on a set of special root servers
  - Malicious nodes can break lookup routing in the circle
- Seems promising, but haven't been used so much
  - Developers tend to use their own proprietary schemes

Names for User, Then What?

- Of course, all objects beyond peer identities need to be somehow named
  - I.e., a file object name could be constructed from the original name and node identifier, or just be the original file name
  - Depends on the application and needs
- Let's pretend that we managed to name our peers, objects and services, what comes next?

Finding Things

- Traditional searching: send query to a search engine
  - A collection of server storing huge amount of data indexed from the Internet
  - Crawlers fetch data for engines, but revise it slowly and cannot access peer systems
- P2P search: peers can conduct the search itself
  - Servers containing search engine database can communicate as peers
  - Database distributed and replicated thus eliminating single-point-of-failure risk

Search and Discovery

- Searching: want to know where *something* is
  - Typical in file sharing, “Metallica I Disappear mp3”
- Discovering: want to know *what* is out there
  - “Who's currently connected in my P2P network?”
- In P2P environment peers need to discover:
  - Resources: what is out there that my applications needs?
  - Presence: who is connected? Who is a member?
  - Content: what are topics discussed in this group?

Resource Discovery

- Classified into two categories
- Discovery in structured P2P systems
  - Based on DHT (Chord, Pastry, Tapestry Kademlia, etc.)
  - Cost of maintaining a consistent distributed index is too high in the dynamic Internet environment
- Discovery in unstructured P2P systems
  - Data placement not controlled (Gnutella, Freenet, etc.)
  - More resilient in dynamic environments
  - High search overhead or massive network traffic

P2P Searching

- Difference between searching and discovery becomes blurred with more sophisticated searches
  - E.g. OpenCola content delivery system adds “similarity” and scores “relevance” of our search
  - “What relevant documents other peers have?”, search becoming discovery
- Standard protocols can be applied to P2P searching
  - UDDI: register and publish any service or resource
  - LDAP: interfacing directories, distributed and secure

Connectivity

- P2P harnesses resources at the edge of the net
  - Rely components over which we have little or no control
  - Nodes exercise local autonomy and join and leave network as they please
  - Found content and resources can disappear unexpectedly
- Connectivity in P2P environment involves various topics
  - Let's identify the problems and present example solutions

Intermittent Presence

- Affects interaction with the peer and message routing
- In Magi by Endeavors: handled by some event and cache services
  - Keep undelivered message in outbox, send when recipient peer is connected
  - Route messages trough another peer (store-and-forward)
  - Route through a central store-and-forward (server)
  - Undelivered messages discarded according to a “TTL”

Fault-tolerance

- Large distributed systems: node failures common
- How to recover and continue from a failure?
- Avaki: a computing resource sharing P2P system
  - Present all entities as objects
  - Atomically save state of all core objects
    - Object can be returned from a crash failure
  - Replicate states onto multiple storage devices
  - Replicate critical object classes
    - Non-available resource request routed to an equivalent service

Content Coherency and Synchronisation

- Ensuring that all peers access the same content
- Groove (Internet based collaboration): shared business space where users place their content
  - When members are offline, online “sending” members forward their changes to a relay service
  - Relay service transmits changes to returned members
  - Offline user can continue working (queue changes to relay)
  - In general, each peer has all content all the time, subject to synchronisation after being disconnected

Availability of Resources

- Closely related to intermittent connectivity
- Entropia: P2P distributed computing
  - Peers execute tasks that lasts from minutes to hours
  - If connectivity restored before completion, just proceed
  - If the disruption lasts longer than estimated completion, relevant jobs restarted at a different peer host
  - Jobs can be replicated for aggregation or security reasons
  - Resource availability for a peer is added when the peer connects server, and removed due to disconnect

Availability of Content

- Improve availability of remote content
- WebRiposte (Data Management and distributed messaging)
  - A multi-level replicated cache shared by peers
  - When a peer requests an item, it's available to others via caching mechanism
  - These single shared local caches are synchronised to correspondence servers providing the upper level cache
  - Distributing content ahead of time to requesting peers

Fault-tolerant Use of Centralised Servers

- Relay, indexing, etc. servers produce a single point of failure risk – usually multiple servers
- “Virtual neighbours” in WebRiposte
  - Refers to a list of correspondence servers that peer holds
  - Peer picks one server and all traffic sent only to it
  - Peer can fail-over to another “virtual neighbour”
  - Correspondence servers virtual neighbour for many peers
  - Random assignment guarantees availability and load balancing

Connectivity: Lessons and Observations

- The more transparencies we have, the more usable the P2P network is
- Users, content and objects move around
  - Don't care where they are located, or if they are replicas
- Most transparencies have to do with availability and fault-tolerance
  - For a conclusion, list the most important ones

Transparencies

- Access
  - The mechanism used for a local call is the same as for a remote call
- Location
  - The caller does not need to know where the object is located
- Failure
  - If the object fails, the caller is unaware: somehow the service recovers and the request is performed

Transparencies

- Heterogeneity
  - Architecture and OS boundaries are invisible
- Migration
  - The caller does not need to know whether an object has moved since they last communicated with it
- Replication
  - Is there one or many object behind a name? The caller does need to know or deal with coherence issues

Transparencies

- Concurrency
  - Are there other concurrent users of an object? The caller does not need to be aware of them
- Scaling
  - Increase or decrease in the number of servers requires no change in the software. Performance may vary
- Behavioural
  - Whether an actual object or a simulation of the object is used is irrelevant (“Am I talking to a real or virtual host”)

Resource Management

- Typically deals with content (files), storage (disk space) and transmission capabilities (bandwidth)
  - Minimum is insertion and location
  - Additionally management facilities for content removal, update and version control, managing storage space and limiting bandwidth
- Different levels of managements depending on the applications and their needs
  - Some general content sharing issues discussed next

Content Management

- Content deletion and update
  - The simplest: immutable files, update is always a new file
  - Preventing unpublishing (removal) may also serve as protection against attacks or censorship
  - Freenet removes least popular files by heuristics
- Content expiration
  - In FreeHaven system, files expire according to a *contract* made between the peers

Content Management Cont'd

- Content versioning
  - OceanStore offers version-based archival system: files in permanent read-only format spread over hundred of nodes, document Id retrieves most recent version along with the history
- Directory structure
  - Mnemosyne system: an entire distributed directory structure built on top of files
  - Includes Unix-like inodes, directories can be used as a common key pointing to a set of files

Content Searching

- Unstructured networks (Gnutella, Kazaa, FreeHaven) have keyword based search mechanism
  - Convenient for user, but difficult to scale well
- Structured networks (Chord, CAN, Pastry) provide only addressing via file identifiers
  - Efficiency, but requires knowledge of exact identifier
- Keyword based search on top of exact-match?
  - Freenet uses indirect files, published along with regular files, named according to relevant search keywords

Storage and Bandwidth Management

- Usually peer can specify the distributed disk space
- Is there need for encryption?
- Management through some economy or resource-trade mechanism
- Denial-of-service attacks: control somehow publishing new files
  - Solve computational work, per-node limit rates
- Kademlia: trade bandwidth for lower latency lookups