University of Babylon
College of Information Technology
Dept. of Information Networks
Data Communication and Networking I.
4^{th} Stage.

Lecture 4

Peer to Peer Networks

1 Introduction

In a client-server architecture, there is an always-on host, called the server, which services requests from many other hosts, called clients. A classic example is the Web application for which an always-on Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host. Note that with the client-server architecture, clients do not directly communicate with each other; for example, in the Web application, two browsers do not directly communicate. Another characteristic of the client-server architecture is that the server has a fixed, well-known address, called an IP address. Because the server has a fixed, well-known address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address. Some of the better-known applications with a client-server architecture include the Web, FTP, and e-mail. The client-server architecture is shown in Figure 1- (a).

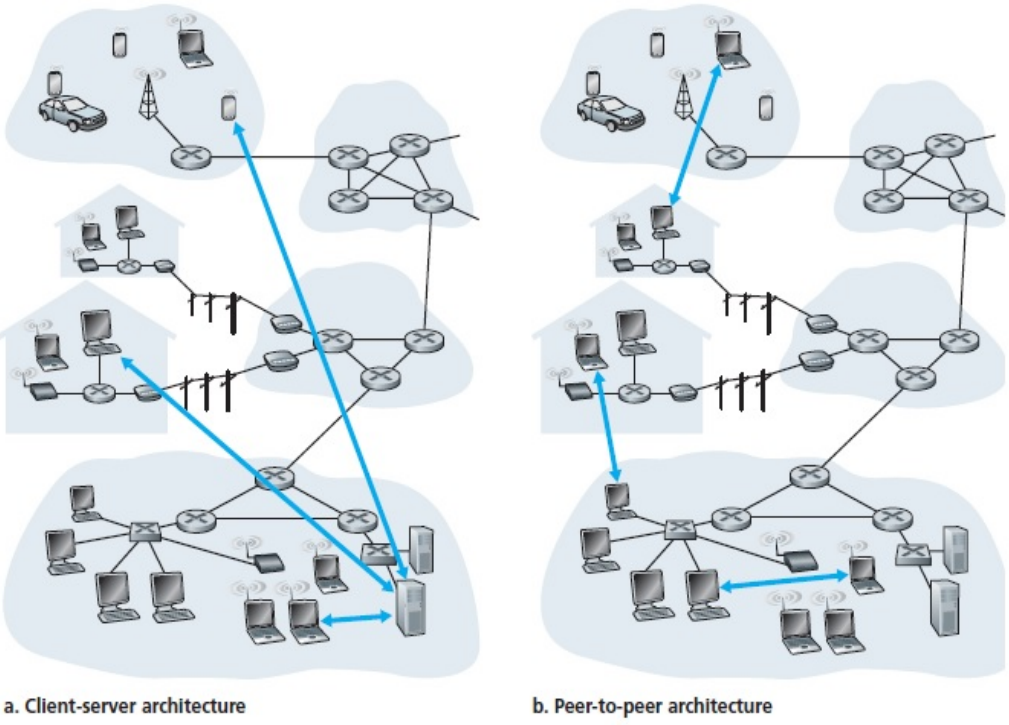

Figure 1: (a) Client-server architecture; (b) P2P architecture.

In a P2P architecture, there is minimal (or no) reliance on dedicated servers in data centers. Instead the application exploits direct communication between pairs of intermittently connected

hosts, called peers. The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices. Because the peers communicate without passing through a dedicated server, the architecture is called peer-to-peer. Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g., BitTorrent), peer-assisted download acceleration (e.g., Xunlei), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream). The P2P architecture is illustrated in Figure 1 (b). Note that some applications have hybrid architectures, combining both client-server and P2P elements. For example, for many instant messaging applications, servers are used to track the IP addresses of users, but user-to-user messages are sent directly between user hosts (without passing through intermediate servers). One of the most compelling features of P2P architectures is their self-scalability. For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers. P2P architectures are also cost effective, since they normally don't require significant server infrastructure and server bandwidth (in contrast with clients-server designs with datacenters). However, future P2P applications face three major challenges:
1. **ISP Friendly:** most residential ISPs (including DSL and cable ISPs) have been dimensioned for "asymmetrical" bandwidth usage, that is, for much more downstream than upstream traffic. But P2P video streaming and file distribution applications shift upstream traffic from servers to residential ISPs, thereby putting significant stress on the ISPs. Future P2P applications need to be designed so that they are friendly to ISPs.
2. **Security:** because of their highly distributed and open nature, P2P applications can be a challenge to secure.
3. **Incentives:** the success of future P2P applications also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications, which is the challenge of incentive design.

2 Advantages and Disadvantages of P2P paradigm

The main advantages of P2P paradigm are:

- Efficient use of resources:
  -Unused bandwidth, storage, processing power at the edge of the network
- Improved scalability:
  -Consumers of resources also donate resources.
  -Aggregate resources grow naturally with utilization.
- Reliability:
  -Replicas.
  -Geographic distribution.
  -No single point of failure.
- Ease of administration:
  -Nodes self organize.
  -No need to deploy servers to satisfy demand (c.f. scalability).
  -Built-in fault tolerance, replication, and load balancing.

Whereas the main disadvantages of P2P networks are:

- Weak security.
- Lack of centralized control. Computers with shared resources may suffer from lazy performance.

3 P2P Properties

The following properties are characteristics found in most P2P systems:

- Unreliable, uncoordinated, unmanaged:
  - No central authority, peers are completely independent.
  -Increases flexibility of individual peers, but makes the overall system (possibly) unreliable.
- Resilient to attacks, heterogeneous:
  -Large number of peers in the system, hard to bring it down?
  -Heterogeneous peers make viruses and worms harder to write?
- Large collection of resources:
  -Voluntary participation, global reach.
  -Millions of simultaneous users.

4 P2P Case Study: BitTorrent

We begin our investigation into P2P by considering a very natural application, namely, distributing a large file from a single server to a large number of hosts (called **peers**). The file might be a new version of the Linux operating system, a software patch for an existing operating system or application, an MP3 music file, or an MPEG video file. In client-server file distribution, the server must send a copy of the file to each of the peers placing an enormous burden on the server and consuming a large amount of server bandwidth. In P2P file distribution, each peer can redistribute any portion of the file it has received to any other peers, thereby assisting the server in the distribution process. The most popular P2P file distribution protocol is BitTorrent.
BitTorrent is a popular P2P protocol for file distribution. In BitTorrent lingo, the collection of all peers participating in the distribution of a particular file is called a **torrent**. Peers in a torrent download equal-size chunks of the file from one another, with a typical chunk size of 256 KBytes. When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or (altruistically) remain in the torrent and continue to upload chunks to other peers. Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent. Let's now take a closer look at how BitTorrent operates. Since BitTorrent is a rather complicated protocol and system, we'll only describe its most important mechanisms. Each torrent has an infrastructure node called a **tracker**. When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent. In this manner, the tracker keeps track of the peers that are participating in the torrent. A given torrent may have fewer than ten or more than a thousand peers participating at any instant of time. As shown in Figure 2, when a new peer, Alice, joins the torrent, the tracker randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers, and sends the IP addresses of these 50 peers to Alice. Possessing this list of peers, Alice attempts to establish concurrent TCP connections with all the peers on this list. Let's call all the peers with which Alice succeeds in establishing a TCP connection "neighboring peers". (In Figure 2, Alice is shown to have only three neighboring peers. Normally, she would have many more.) As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections with Alice. So a peer's neighboring peers will fluctuate over time. At any given time, each peer will have a subset of chunks from the file, with different peers having different subsets. Periodically, Alice will ask each of her neighboring peers (over the TCP connections) for the list of the chunks they have. If Alice has L different neighbors, she will obtain L lists of chunks. With this knowledge, Alice will issue requests (again over the TCP connections) for chunks she currently does not have. So at any given instant of time, Alice will have a subset of chunks and will know which chunks her neighbors have. With this information, Alice will have two important decisions to make. First, which chunks should she request first from her neighbors? And second, to which of her neighbors should she send requested chunks? In deciding which chunks to request, Alice uses a technique called **rarest first**. The idea is to determine, from among the

chunks she does not have, the chunks that are the rarest among her neighbors (that is, the chunks that have the fewest repeated copies among her neighbors) and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to (roughly) equalize the numbers of copies of each chunk in the torrent.

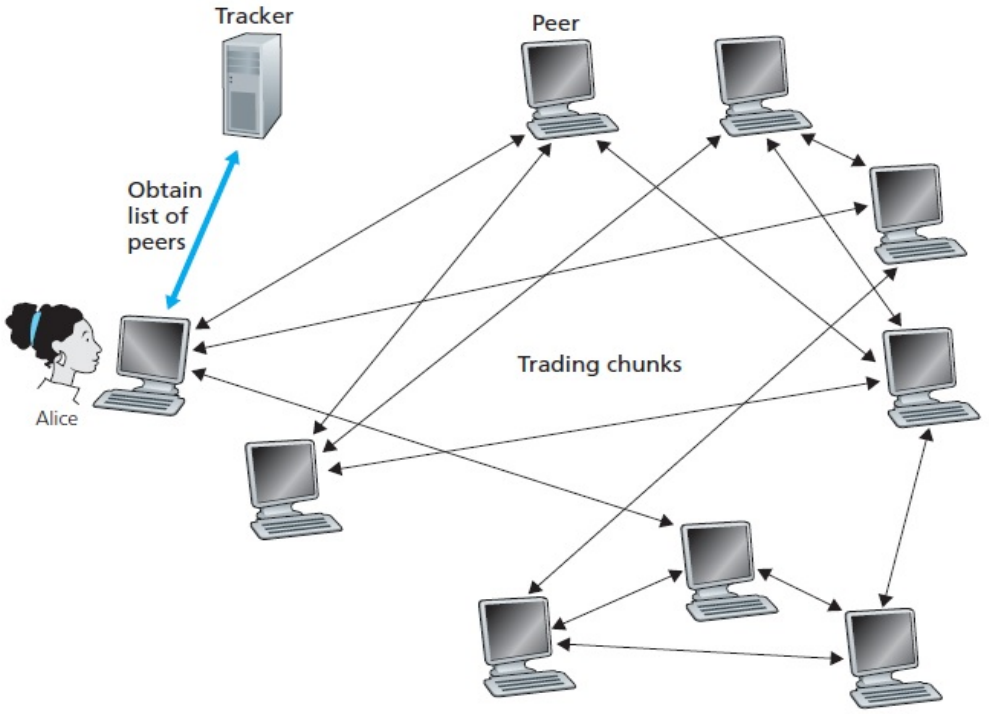

Figure 2: File distribution with BitTorrent.

To determine which requests she responds to, BitTorrent uses a clever trading algorithm. The basic idea is that Alice gives priority to the neighbors that are currently supplying her data at the highest rate. Specifically, for each of her neighbors, Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate. She then reciprocates by sending chunks to these same four peers.