# Peer to peer (III)

**Norberto Fernández**
**Departamento de Ingeniería Telemática**
**http://www.it.uc3m.es/berto/**



# Tablas Hash Distribuidas (DHT, *Distributed Hash Tables*)

# Introducción

- Conjunto de *peers* que se comportan como una tabla *hash*
- Principios fundamentales:
  - Se define un espacio de direccionamiento
  - A cada unidad de datos (Ej.: fichero) se le asigna un identificador en el espacio de direccionamiento
    - Ej.: Mediante *hash* de su nombre o su contenido
  - Cada nodo de la red es responsable de un fragmento del espacio de direccionamiento



# Introducción

- Operaciones básicas sobre la tabla:
  - **put**(ID, datos)
    - Localizar al nodo responsable del fragmento del espacio de direccionamiento que contiene a ID y almacenar en él el contenido
  - datos **get**(ID)
    - Localizar al nodo responsable y solicitarle el contenido
- Adicionalmente se definen mecanismos para gestionar la entrada/salida de nodos de la red
  - Transferencia de la responsabilidad sobre parte del espacio de direccionamiento

# Introducción

- Buenas propiedades de balanceo de carga
  - Si la función de *hash* se escoge apropiadamente
    - Se tiende a repartir los contenidos uniformemente entre los nodos
- Sistema distribuido
  - Ventaja sobre Napster
- Las consultas se encaminan hacia el nodo responsable del fragmento del espacio de direcciones asociado
  - No es necesario *flooding* como en Gnutella
- Si un contenido está en la red se garantiza su localización
  - En Gnutella podía pasar que no se encontrase



# Introducción

- Problema: difícil llevar a cabo consultas complejas
  - Búsqueda por IDs
- Múltiples variantes sobre esta misma idea básica:
  - CAN, Chord, Pastry, Tapestry, Kademlia, etc.
    - Diferencias espacio de direccionamiento, mecanismos de encaminamiento de consultas, etc.
- Algunas aplicaciones:
  - Sistemas de correo P2P (Ej.: ePOST)
  - Compartir ficheros (Ej.: eMule -Kademlia-)

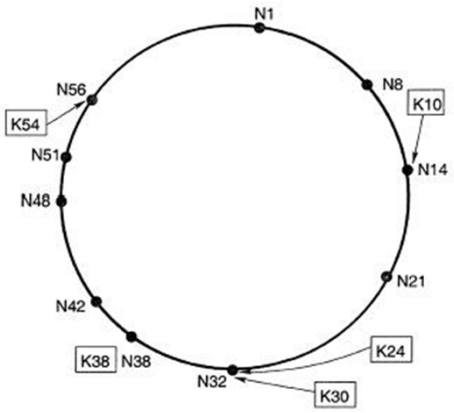
# Chord



# Introducción

- Desarrollado en el MIT
  - Stoica et al., 2001
- Espacio de claves: L bits
  - Enteros en el rango [0, 2^L – 1]
  - Topología circular
- Cada nodo tiene un **identificador** en el espacio de claves
- Cada unidad de datos (Ej.: fichero) tiene una **clave** en el espacio de claves
- La unidad de datos de clave K se almacena en el nodo de identificador ID ≥ K

# Espacio de claves (L=6 bits)




Fuente: Stoica et al. 2001



# *Routing* (I)

- Cada nodo mantiene una tabla de rutas que permite localizar a otros nodos (*finger table*)
  - Se mantiene también una referencia al predecesor en el anillo
- En el nodo de ID=N la fila i-ésima de la tabla de rutas almacena la información sobre el nodo que almacenaría la clave de valor:

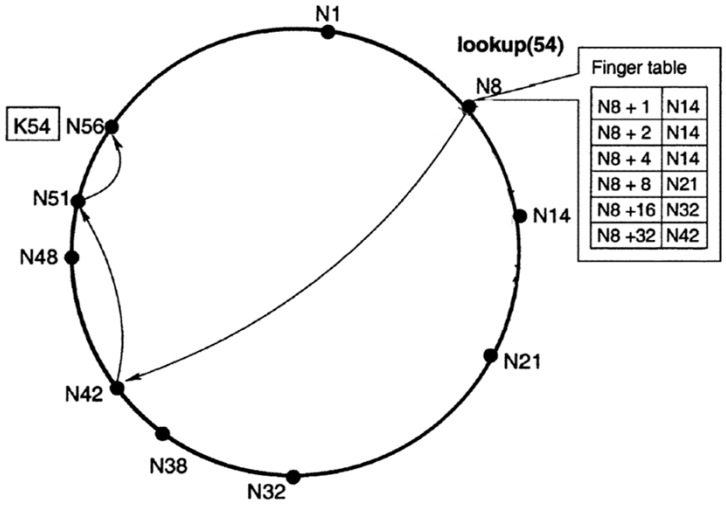
  N+2^(i-1)
- Si el espacio de claves tiene tamaño L, el tamaño máximo de la tabla de rutas es L
  - Independiente del número de nodos en la red

# *Routing* (II)

- Un nodo envía la consulta sobre la clave K al nodo cuya información se encuentre en su tabla de rutas, cuyo ID < K y que esté lo más cerca posible de K
  - Si ese no responde, se la envía al siguiente más próximo, etc.
- Cuando una consulta sobre la clave K llega a un nodo tal que K queda entre el valor de su ID y el valor del ID de su sucesor en el círculo, el nodo responde indicando el ID de su sucesor en el círculo



# *Routing* (III)




Fuente: Stoica et al. 2001

# Manteniendo la red Chord (I)

- Entrada de nodos:
  - Se genera un ID para el nuevo nodo (Ej.: aleatorio)
  - Necesario conocer a un nodo que ya esté en la red
  - Se consulta a ese nodo por la clave K=ID
    - Se obtiene una referencia al sucesor en el anillo
  - Se avisa al sucesor para que actualice la información sobre su predecesor y diga al nuevo nodo cuál es su predecesor



# Manteniendo la red Chord (II)

  - Se contacta al sucesor para que transfiera los datos de los que se debe responsabilizar el nuevo nodo
  - Se genera la tabla de rutas del nuevo nodo consultando sucesivamente a la red por las claves: ID+2^(i-1) i=2…L
- Salida de nodos:
  - Le pasan a su sucesor la responsabilidad de su fragmento del espacio de claves e información sobre su nuevo predecesor
  - Avisan al predecesor de cuál es su nuevo sucesor

# Manteniendo la red Chord (III)

- Entradas concurrentes/caída de nodos:
  - Los nodos que entran en la red no obtienen el predecesor directamente: Protocolo de estabilización
    - Cada nodo pregunta periódicamente a su sucesor acerca de cuál es su predecesor
  - Actualización periódica de las tablas de rutas
    - Se hace una consulta para comprobar si el nodo asociado a la fila i-ésima ha cambiado
  - Si un nodo no responde al contactarlo, se cambia su entrada en la tabla de rutas
  - Si un nodo se cae, sus datos dejan de estar disponibles: replicación de datos en sucesor



# Referencias


- The Gnutella Protocol Specification v0.4:
  - http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
- BitTorrent Protocol Specification v1.0 (Theory.org):
  - http://wiki.theory.org/BitTorrentSpecification
- BitTorrent Protocol Specification (Oficial):
  - http://www.bittorrent.org/protocol.html
- [Stoica et al. 2001] Chord: A Scalable Peer to peer Lookup Service for Internet Applications
  - http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

# Bibliografía

- **Peer-to-Peer Systems and Applications**
  - Ralf Steinmetz, Klaus Wehrle (Eds.)
  - Springer, 2005. ISBN: 3-540-29192-X


- **Peer-to-peer: Harnessing the Benefits of a Disruptive Technology**
  - Andy Oram (Ed.)
  - O'Reilly, 2001. ISBN: 0-596-00110-X