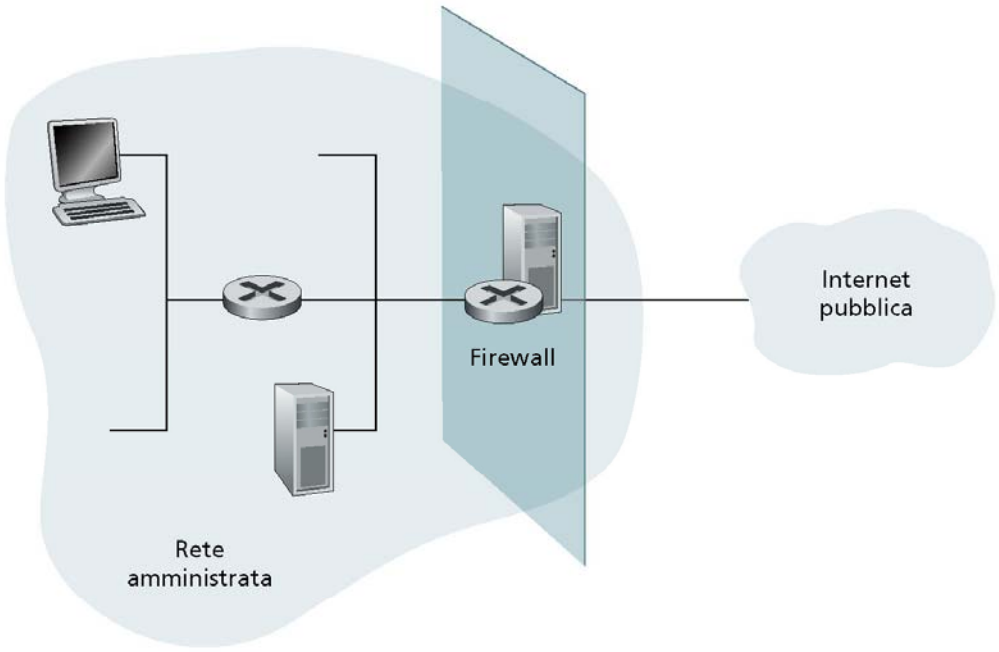
Elementi sull'uso dei firewall

Carlo Mastroianni

Firewall

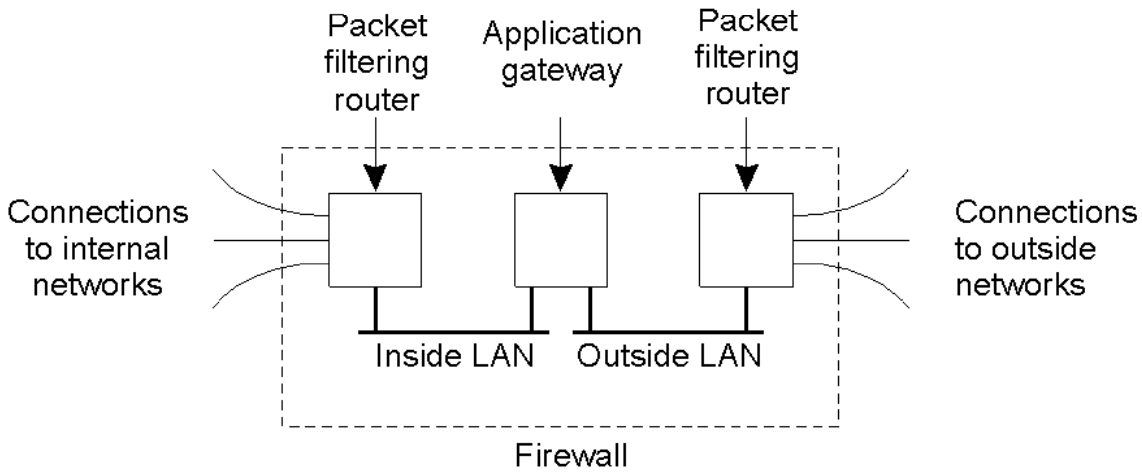
- Un **firewall** è una combinazione di hardware e software che protegge una sottorete dal resto di Internet
- Il firewall controlla l'accesso tra le risorse interne ed il mondo esterno, permettendo ad alcuni pacchetti di passare, e bloccandone altri
- Ci sono due tipi di firewall:
  - **Firewall a filtraggio di pacchetto**, che operano nello strato di rete
  - **Gateway a livello di applicazione**

Uso di un firewall



Firewall "completo"

- Per aumentare la sicurezza, spesso si utilizzano due firewall a filtraggio di pacchetto, ed in mezzo un gateway a livello delle applicazioni



Filtraggio di pacchetto

I filtri di pacchetto esaminano le intestazioni IP e TCP/UDP, ed applicano delle regole di filtraggio basate su:

- **Indirizzi IP** di sorgente e di destinazione
- **Porte** sorgente e destinazione di TCP e UDP
- Tipo di messaggio **ICMP** (es. ping è spesso disabilitato)
- Segmenti TCP di **inizializzazione** della connessione
  - Il primo segmento di una connessione TCP ha il bit ACK impostato a 0, tutti gli altri hanno ACK=1
  - Filtrando i segmenti in arrivo con ACK=0 si bloccano tutte le connessioni originate all'esterno della sottorete, mentre sono consentite quelle originate all'interno

Filtraggio di pacchetto: esempio

- Alice amministra la rete di una società 222.22.0.0/16 (maschera 255.255.0.0) e in generale non vuole consentire l'accesso alla sua rete dall'esterno
- Alice però collabora con l'università di Bob e vuole consentire a Bob ed ai suoi colleghi, che stanno sulla rete 111.11.0.0/16, di accedere esclusivamente alla sottorete interna 222.22.22.0/24
- Tuttavia Alice è a conoscenza che Trudy, una nota hacker, lavora nella sottorete dell'università 111.11.11.0/24 di Bob, e vuole bloccare l'accesso da tale sottorete all'intera rete della società

Filtraggio di pacchetto: esempio

- Le regole descritte possono essere specificate nel firewall come segue:

Regola	Rete sorg.	Rete dest.	Azione
R1	111.11.0.0/16	222.22.22.0/24	permit
R2	111.11.11.0/24	222.22.0.0/16	deny
R3	0.0.0.0/0	222.22.0.0/16	deny

- Tuttavia **l'ordine** delle regole è importante: al pacchetto in arrivo viene applicata la prima regola che è compatibile con gli indirizzi sorgente e destinazione del pacchetto
- In generale conviene mettere prima le regole *deny*, poi quelle *permit*, e poi una regola *deny* di default

Ordinamento delle regole di filtraggio

- Verifichiamo cosa succede se mettiamo le regole nell'ordine R1-R2-R3, oppure nell'ordine R2-R1-R3.

Pacchetto	Indirizzo mittente	Indirizzo destinazione	Azione desiderata	Ordine R1,R2,R3	Ordine R2,R1,R3
1	111.11.11.1	222.22.6.6	deny	**deny (R2)**	**deny (R2)**
2	111.11.11.1	222.22.22.2	deny	***permit (R1)***	**deny (R2)**
3	111.11.6.6	222.22.22.2	permit	**permit (R1)**	**permit (R1)**
4	111.11.6.6	222.22.6.6	deny	**deny (R3)**	**deny (R3)**

IP spoofing

- L'**IP spoofing** è una tecnica spesso utilizzata dagli hacker per effettuare attacchi
- Consiste nel mettere nel pacchetto IP un indirizzo sorgente fasullo
- L'uso dell'IP spoofing rende difficile identificare l'indirizzo IP del computer che ha iniziato l'attacco
- Gli attacchi sono in genere di tipo **Denial Of Service** (**DoS**) e tentano di sovraccaricare un dato server impedendogli di operare efficacemente

Attacco di tipo SYN flooding

- L'attaccante inonda un server di pacchetti **SYN** (pacchetti di inizio connessione aventi bit SYN=1 e bit ACK=0) aventi **indirizzi IP sorgente camuffati**
- Il server risponde con pacchetti **SYNACK** (aventi bit SYN=1 e bit ACK=1) inviati agli host cui corrispondono gli indirizzi IP camuffati
- Tali host ovviamente non completano il 3-way handshake, e la connessione non verrà instaurata
- Tuttavia il server è costretto a creare i socket e ad allocare le rispettive strutture dati, consumando così memoria e risorse del processore

Attacco di tipo Smurfing

- L'attaccante invia un gran numero di pacchetti **ICMP** di tipo **Echo Request** (es. col programma **Ping**) ad un ampio numero di host
- L'indirizzo sorgente di tali richieste è un indirizzo IP camuffato che corrisponde all'host che si vuole attaccare
- I pacchetti ICMP di tipo **Echo Reply** saranno quindi inviati a tale host che sarà quindi sottoposto ad un bombardamento che ne può pregiudicare le funzionalità
- L'attacco è ancora peggiore se i pacchetti ICMP sono inviati ad un **indirizzo multicast** di una rete locale (es. l'indirizzo 150.145.1.255 della rete 150.145.1.0/24): tutti gli host della rete locale risponderanno allo stesso host sotto attacco

Regole anti "spoofing"

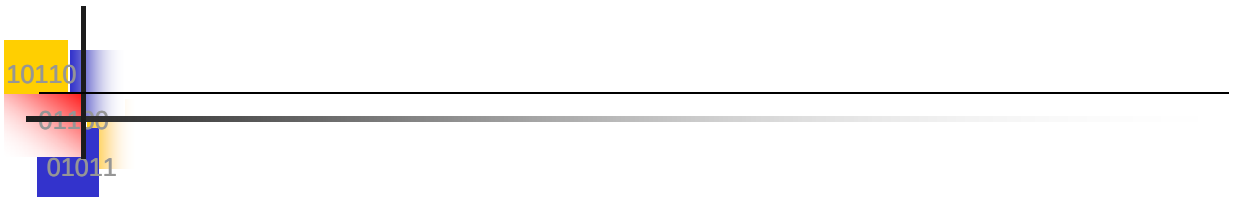
- Per limitare questi attacchi bisogna assicurarsi che i **pacchetti in uscita** dalla rete non abbiamo indirizzi sorgente **diversi** da quelli compresi nel range assegnato alla rete stessa
- Allo stesso modo bisogna bloccare i **pacchetti in entrata** che hanno indirizzi sorgente **appartenenti** al range della rete interna
- Inoltre bisogna bloccare i pacchetti in entrata aventi indirizzi sorgente appartenenti a reti riservate per il **NAT** (reti 10.0.0.0/8, 172.16.0.0/12 e 192.168.0.0/16)
- Gli indirizzi NAT sono indirizzi utilizzabili solo all'interno di una rete locale (es. una rete connessa all'esterno tramite un router ADSL) ma non visibili all'esterno

Altre regole generali di sicurezza

- Per limitare gli attacchi di tipo **smurfing**, un'altra buona regola è non accettare pacchetti in entrata aventi come indirizzo destinazione un **indirizzo multicast**
- È utile inoltre impedire l'accesso di segmenti **TCP di inizio connessione** (aventi ACK=0), in modo da consentire solo le connessioni originate all'interno
  - Non esiste una simile possibilità di controllo con **UDP**: per questo in genere il traffico UDP viene impedito, in toto, o solo per uno specifico set di porte
- Può essere necessario bloccare il traffico relativo a servizi "pericolosi" per vari motivi (es. NFS che consente di accedere a cartelle condivise di un computer remoto) e consentire solo il traffico relativo a servizi importanti (dns, http, ftp, ssh, smtp, pop3, imap ecc.)

Gateway a livello delle applicazioni

- Cosa succede se si vuole consentire una connessione Telnet solo ad alcuni utenti interni, a prescindere dall'indirizzo IP che usano? E se si vuole che tali utenti si autentichino con user e password?
- il firewall a filtraggio di pacchetto non può gestire questa esigenza
- Solo un gateway a livello delle applicazioni può esaminare i dati specifici dell'applicazione, in questo caso di Telnet
- Più gateway, relativi a diverse applicazioni, possono risiedere sullo stesso host
- I server di posta che "esplodono" le mailing list ed i Web proxy sono esempi di gateway delle applicazioni

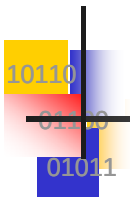


Elementi sui sistemi P2P

Carlo Mastroianni

Sistemi P2P

- In una rete Peer-To-Peer i computer condividono risorse e servizi e li scambiano direttamente tra loro.
- A differenza dell'approccio Client/Server, chi richiede la connessione non è ad un livello gerarchico inferiore: i partecipanti sono dei pari (**peer**).
- Pertanto un peer può agire sia da client sia da server.



Sistemi P2P: caratteristiche

- Perché un sistema si possa definire Peer-to-Peer deve presentare almeno quattro caratteristiche:

1. **peer discovery**: i peer possono ricercare altri peer sulla rete
2. **resource sharing**: i peer condividono le risorse con altri peer
3. **resource discovery**: i peer possono effettuare ricerche ed interrogazioni per scoprire le risorse messe a disposizione da altri peer
4. **dinamicità**: il sistema deve funzionare anche se i peer si disconnettono e riconnettono più o meno frequentemente

P2P puro ed ibrido

- **P2P puro:**

  Tutti i peer hanno contemporaneamente funzioni di client e di server senza nessun mediatore o server centralizzato (**Gnutella**).

- **P2P ibrido:**

  Esistono dei server che hanno funzioni di coordinamento, ma i peer si connettono tra di loro autonomamente (**Napster**, **Emule**).

Sistemi P2P strutturati e non strutturati

- **Sistemi P2P non strutturati:**

  La locazione delle risorse *non è legata* alla topologia della rete. Ogni peer pubblica le risorse autonomamente, quindi non c'è modo di sapere quale peer potrebbe possedere una risorsa (**Napster**, **Gnutella**, **Morpheus**, **Emule**)

- **Sistemi P2P strutturati:**

  La locazione delle risorse *è legata* alla topologia della rete. Esiste un legame tra l'identificativo di una risorsa e l'indirizzo del peer che possiede tale risorsa. È quindi possibile effettuare ricerche mirate delle risorse (**Chord**, **CAN, BitTorrent, KAD, Freenet**)

Gnutella

- E' un software creato nel 2000 da Justin Frankel e Tom Pepper della società Nullsoft.
- È un sistema P2P **puro non strutturato**.
- Per accedere ai servizi di Gnutella, un nodo deve conoscere l'indirizzo di almeno un altro nodo (fase di start up).
- Esistono dei nodi **cache server** che mantengono gli indirizzi di quanti più nodi possibile.
- In fase di connessione alla rete il peer contatta automaticamente uno di questi cache server, che:
  - provvede ad aggiungere alla comunità il nuovo arrivato.
  - comunica al nuovo peer l'indirizzo di almeno un altro peer.

Gnutella: messaggi

Il protocollo Gnutella definisce 6 tipi di messaggi:

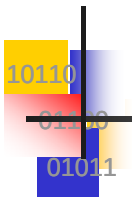
- **Connect**: usato all'inizio per connettersi ad un host della rete Gnutella.
- **Ping**: utilizzato per scoprire altri host sulla rete. Un Servent che riceve un Ping deve rispondere con un Pong se è disponibile ad accettare connessioni.
- **Pong**: utilizzato come risposta al Ping; contiene l'indirizzo del Servent (indirizzo IP e porta) ed informazioni circa i dati condivisi (numero di file, dimensione totale).
- **Query**: utilizzato per effettuare delle richieste di file: il corpo del messaggio contiene una *search string*.

Gnutella: messaggi

- **QueryHit**: la risposta ad una query. Un Servent che riceve una Query deve rispondere con un QueryHit se ha trovato corrispondenza con i dati richiesti. Il messaggio contiene le informazioni per acquisire i dati richiesti tramite HTTP.
- **Push**: se il proprietario del file cercato (es. l'host B) è dietro un firewall che blocca la porta HTTP, il richiedente (l'host A) non può effettuare il download diretto. In questo caso A invia ad B un messaggio di Push e chiede a B di connettersi ad A ed effettuare l'upload del file.

Struttura dell'intestazione del messaggio

Message ID (GUID)	Type of message	TTL	Hops	Payload Length



Gnutella: Fase 1 - Connessione

- Il software del gnode (“servent”) invia un messaggio **Connect** ad un host conosciuto: l’indirizzo di questo host può essere comunicato da un server di cache.
- Per esplorare la rete, il *servent* invia dei pacchetti **Ping**. Ogni macchina che riceve un **Ping**, se disponibile ad accettare connessioni, risponde con un pacchetto **Pong**, che segue la stessa strada, all’indietro, fatta dai pacchetti **Ping**.
- Nei pacchetti **Pong** sono contenute informazioni circa il numero di file condivisi, la loro dimensione, la velocità di connessione ecc.
- I **Pong** ricevuti permettono al Peer di connettersi ad un certo numero di altri Peer, e costituire così un **peer-group** di nodi conosciuti.

Gnutella: Fase 2 - Ricerca

- Una volta che si è entrati a far parte della rete si può effettuare la ricerca di un file.
- Il software *servent* invia in broadcast a tutti i peer conosciuti un messaggio di **Query** con i criteri di ricerca stabiliti dall’utente.
- Ogni peer che riceve una **Query** inoltra a tutti i peer conosciuti lo stesso messaggio.
- Ogni pacchetto viene contraddistinto con un **GUID** (**Global Unified Identificator**), in modo da non propagare lo stesso pacchetto a macchine che lo hanno già ricevuto.

Gnutella: Fase 2 - Ricerca

- In seguito alla richiesta arriveranno al richiedente i pacchetti di **QueryHit** con i risultati e le informazioni circa la velocità di collegamento di chi possiede il file.
- L'utente può quindi scegliere il peer da cui prelevare il file.
- La connessione tra il richiedente ed il donatore è diretta ed è effettuata con un download tramite il protocollo **HTTP**.
- Se il donatore è dietro un firewall, il richiedente invia un messaggio di **PUSH** al donatore, e quest'ultimo effettua l'upload del file verso il richiedente.

Gnutella: problematiche

1) Invio in broadcast dei messaggi

- Comporta un carico di rete eccessivo.
- È compensato con tecniche di vario tipo: es. l'identificazione dei pacchetti di Query tramite il GUID (così da scartare le repliche), l'uso del Time To Live.

2) Sbilanciamento tra i peer

- Ci possono essere peer che offrono molte risorse e peer che si limitano a chiederle. Il peer-to-peer può così degenerare in un sistema client-server.
- È opportuno scoraggiare comportamenti di tipo "parassita".

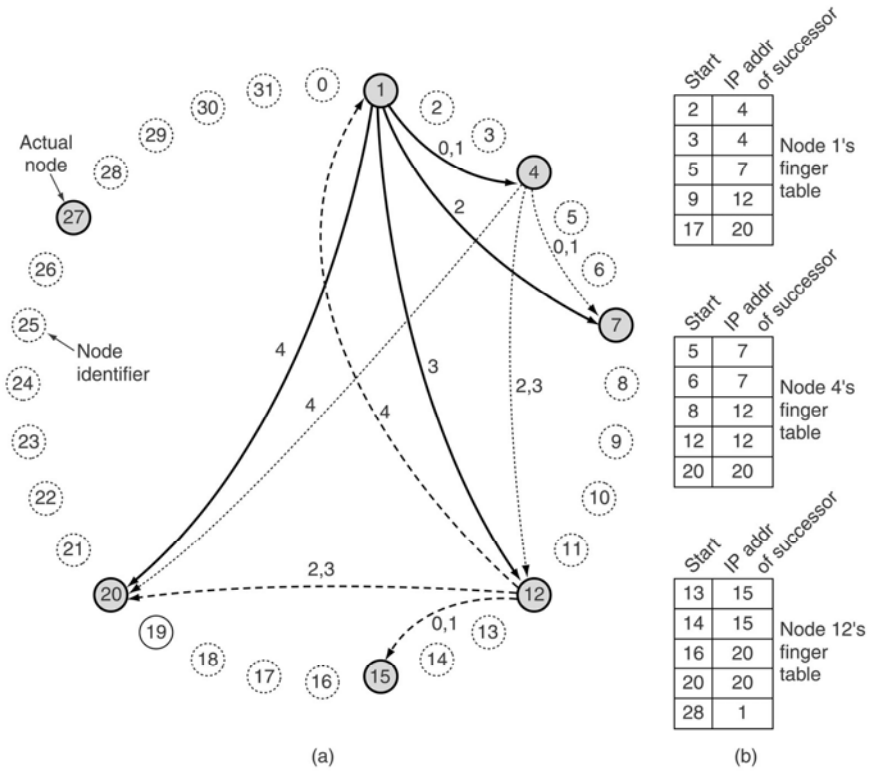
Chord: codici dei peer

- **Chord** è un sistema P2P **puro** e **strutturato**: esiste una precisa associazione tra le risorse ed i peer che le gestiscono.
- Ad ogni peer è assegnato un codice **identificativo con m bit**.
- Ogni peer mette a disposizione delle risorse ed è disposto a gestire alcune informazioni che consentono di indirizzare le ricerche degli altri peer.
- Chord gestisce un cerchio logico che contiene tutti i 2^m codici dei peer, di cui solo alcuni sono assegnati a peer effettivamente presenti nella rete.
- La funzione **successor(K)** identifica il primo peer con codice successivo a K. Ad esempio, nella configurazione mostrata in figura, successor(8) è 12.

Chord: assegnamento delle risorse

- Così come ai peer, anche alle risorse è assegnato un codice. A partire dal nome della risorsa è generata, tramite una funzione di **hash**, una **chiave di m bit**.
- Quando un peer vuole pubblicare una risorsa, affida tale risorsa al peer avente codice uguale alla chiave **C** della risorsa. Se tale peer non è presente, la risorsa è assegnata al peer con codice uguale a **successor(C).**
- Ogni peer deve conoscere l'indirizzo del suo successore.
- Inoltre ogni peer gestisce una **finger table** che contiene gli indirizzi di altri **m** peer:
  - Il record **i-esimo** (i=0..m-1) della finger table del nodo avente indice **K** ha un campo **start** ottenuto sommando 2^i a **K**. Il peer puntato dal record i-esimo è il peer avente codice uguale a **successor(start)**.

Chord: esempio di configurazione



Chord: ricerca delle risorse

1. Quando un peer **K** vuole ricercare una risorsa, per prima cosa calcola la chiave **C** della risorsa cercata, applicando la funzione di hash al nome della risorsa.
2. Poi il peer verifica se la chiave **C** è compresa tra **K** e **successor(K)**: in questo caso la risorsa è certamente gestita da successor(K). Poiché K conosce l'indirizzo di successor(K), la ricerca termina.
3. Altrimenti il peer usa la **finger table** e verifica qual è il peer, puntato dalla tabella, che è *il più vicino predecessore* di **C**. La richiesta è inoltrata a tale peer e riparte da lì. La ricerca termina quando si verifica il caso **2**.

- La ricerca si completa in **log(n)** passi, dal momento che ad ogni passo è possibile dimezzare lo spazio di ricerca.
- **Esempio.** se il peer 1 vuole cercare la risorsa con chiave 14, inoltra la richiesta al peer 12. Il peer 12 si accorge che la chiave 14 è inferiore al codice del suo successore (15), e quindi conclude che la risorsa è posseduta dal peer 15.

Approccio strutturato: vantaggi e svantaggi

- **Vantaggi**
  - Le ricerche sono molto più veloci rispetto ai sistemi P2P non strutturati.
  - Il carico di rete ed il carico di elaborazione sui peer sono inferiori.
- **Svantaggi**
  - Si perde in flessibilità: è possibile ricercare risorse con nomi ben determinati, ma è complicato ricercare risorse che presentino particolari caratteristiche (es. un software che svolga un certo tipo di elaborazione, un PC con velocità della CPU compresa in un dato intervallo).
  - È necessario eseguire una procedura di riassegnazione delle risorse ogni volta che un peer entra nel sistema o lo abbandona. Se il sistema è molto dinamico questo diventa un problema.

Introduction to Cloud computing

Cloud computing

- **Cloud computing** is a style of computing in which **dynamically scalable** and often **virtualized** resources are provided as a service over the Internet.
- Users need not have knowledge of, expertise in, or control over the technology infrastructure in the “Cloud" that supports them.

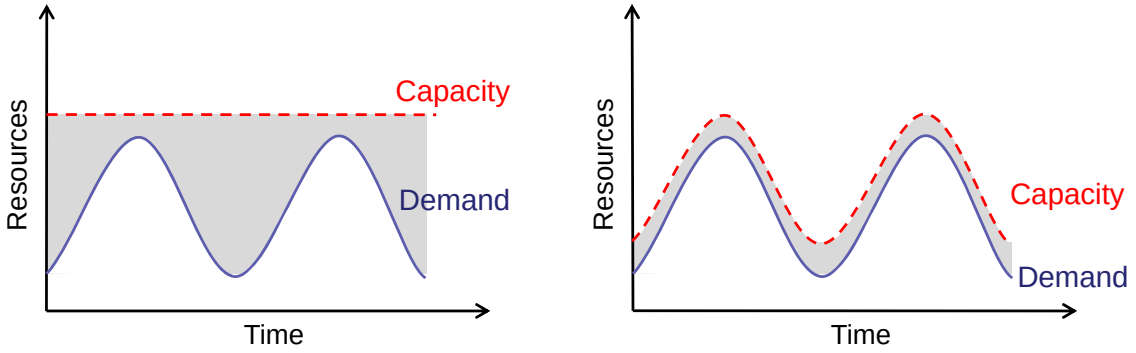
Cloud computing in a few words

- Rent computing resources from a provider
- Dynamically increase or shrink computing units
- Pay only for resources that you actually use
  - Computing power, network bandwidth, storage

From “Running SOA in a grid“, by David Beisert

Economics of Cloud computing

Pay by use instead of provisioning for peak



Static data center

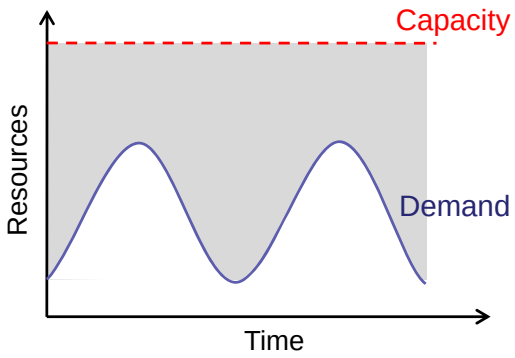
Data center in the cloud

Unused resources

From “Above the Clouds”, by UC Berkeley RAD Lab.

Economics of Cloud computing

Risk of over-provisioning: underutilization

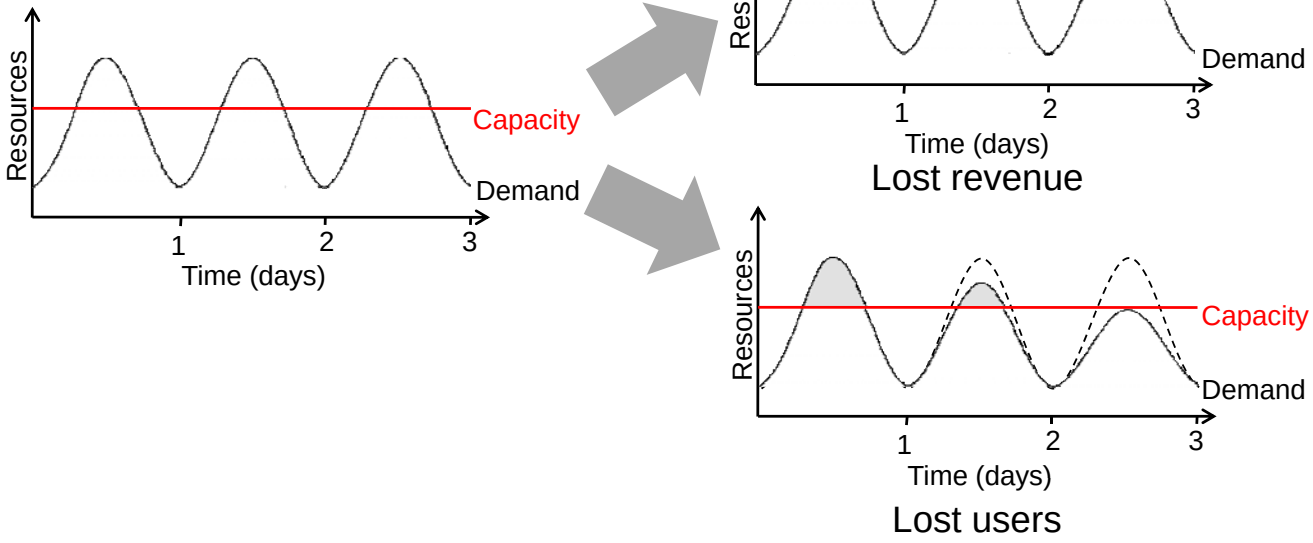


Unused resources

Static data center

Economics of Cloud computing

Heavy penalty for under-provisioning



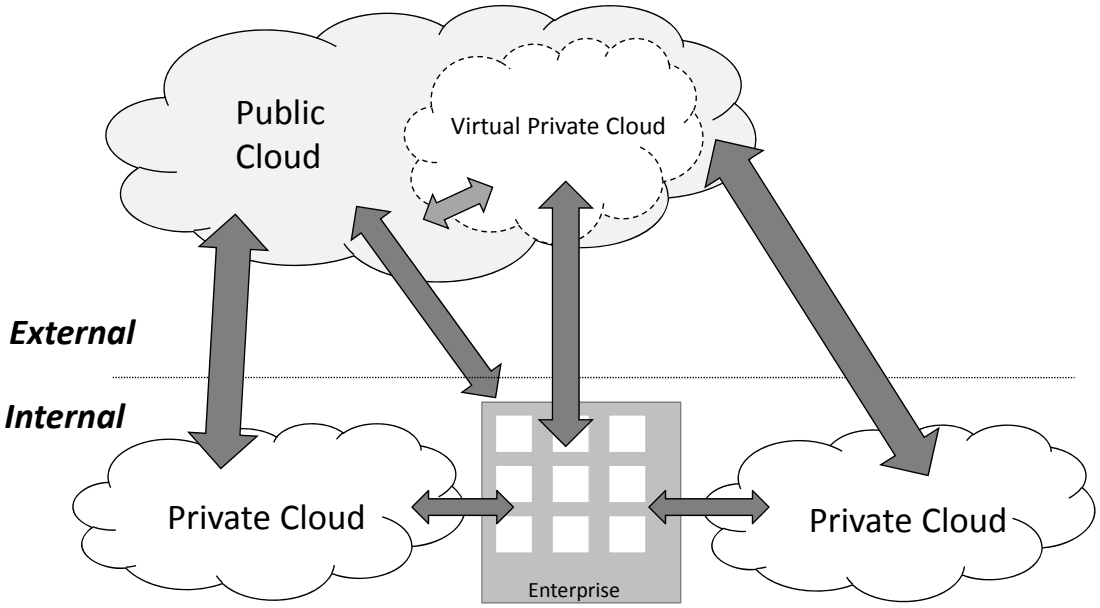
Cloud deployment models

- Private cloud
  - enterprise owned or leased
- Community cloud
  - shared infrastructure for specific community
- Public cloud
  - Sold to the public, mega-scale infrastructure
- Hybrid cloud
  - composition of two or more clouds

http://csrc.nist.gov/groups/SNS/cloud-computing/index.html

From “Cloud Computing and Virtualization: Security and Audit”, by Vern Williams.

Cloud ecosystems



From "Cloud Computing", by Farhad Javidi.

Cloud computing services

- Software as a Service (SaaS)
  - Software is provided to end users in an "On-demand" fashion.
  - Reduces upfront costs, i.e. buying multiple licenses
  - "Utility-based" computing
- Platform as a Service (PaaS)
  - When the software needed to develop Cloud applications are *themselves* provided in a "Software as a Service" fashion
- Infrastructure as a Service (IaaS)
  - An "infrastructure" referring to much of the background *hardware* (contrast to software) needs of an organization

From "Computing on the cloud" by Jason Detchevery.

Cloud and data centers

- Clouds are hosted on data centers
- Size ranges from tens to tens of thousands of physical servers
- Inefficiencies cause:
  - ✓ high electricity costs (also for cooling)
  - ✓ huge carbon emissions
  - ✓ server overload and low QoS
- Data center efficiency is a huge issue!

Facebook data center in Sweden

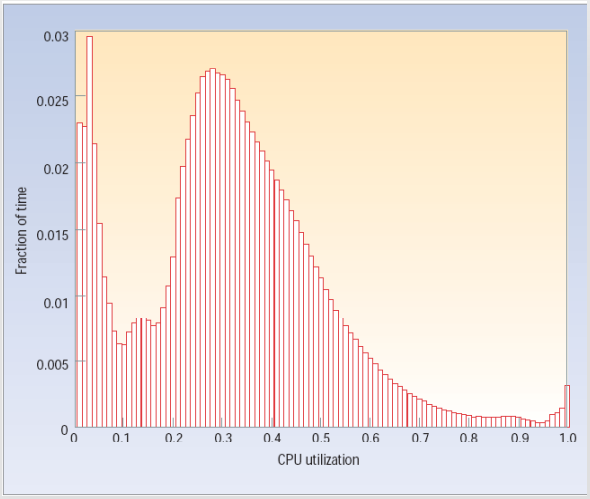
Inefficiency of servers

Two sources of inefficiency

1. On average only 30% of server capacity is exploited
2. Active but low-utilized servers consume more than 50% of the energy consumed when fully utilized

Carlo Mastroianni
Works '12 at Supercomputing 2012, Salt Lake City, November 2012

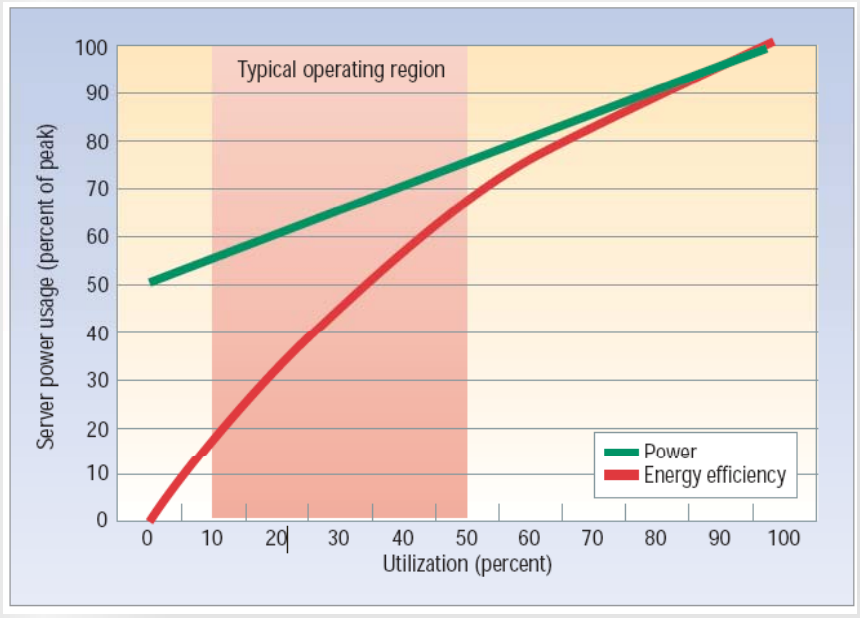
Typical utilization of servers



most servers are in 20% to 40% region of CPU utilization

Source: L.Barroso, U.Holzle, The case of energy proportional computing, ACM Computer Journal, Volume 40 Issue 12, December 2007.

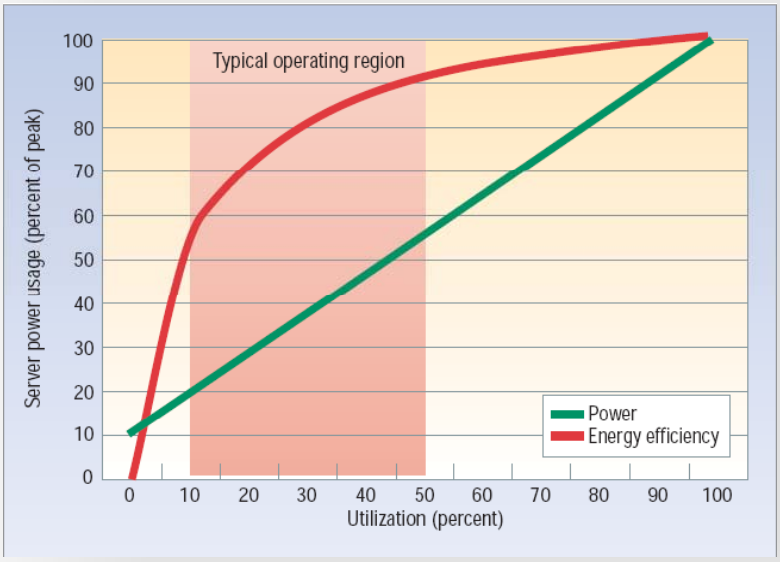
Typical efficiency behavior



Power use is 50% when idle

Energy efficiency is utilization divided by power value

Desired energy efficient behavior



Power is almost proportional to utilization

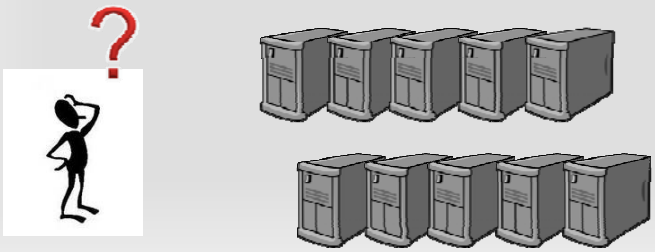
Efficiency is high for typical values of utilization

Current solutions for data centers

- Adopt "energy-efficient" servers
  - e.g., voltage and frequency scaling
  - good for CPU (and partially RAM), not for other components
- Consolidate VMs on servers & hibernate unused servers
- More efficient cooling

Consolidation of VMs in data centers

- Assign VMs on the smallest number of servers, so as to hibernate the remaining servers, and save energy
- An NP-hard problem (online bin packing problem)
- Solutions available today are often complex, not scalable and may require a massive reassignment of VMs



Some current solutions for consolidation

- Best Fit: each VM is assigned to the server whose load is the closest to a target

  this only guarantees a performance ratio of 17/10: at most 17 servers are used when the minimum is 10

- Best Fit Decreasing: VMs are sorted in decreasing order, then assigned with Best Fit

  performance ratio is 11/9, but many concurrent migrations

- DPM of VMWare adopts a greedy algorithm

  servers are sorted according to numerous parameters (capacity, power consumption, etc.)

  DPM scans the list and checks if servers can be unloaded and hibernated

eco4cloud solution

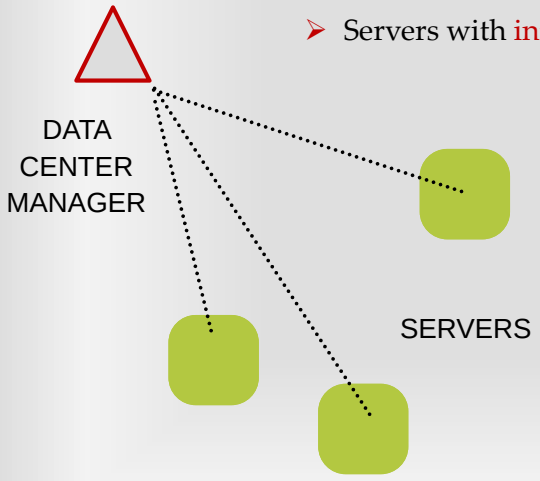
www.eco4cloud.com

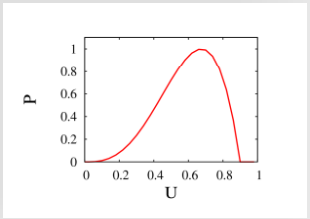
C. Mastroianni, M. Meo, G. Papuzzo, "Self-Economy in Cloud Data Centers: Statistical Assignment and Migration of Virtual Machines", Euro-Par 2011, September 2011.

eco4cloud in a nutshell

The data center manager assigns and migrates VMs to servers based on local probabilistic trials:

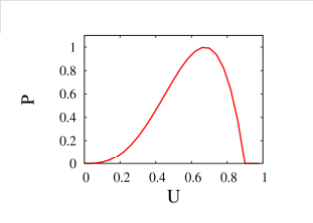
- Lightly loaded servers tend to reject VMs
- Highly loaded servers tend to reject VMs
- Servers with intermediate load tend to accept VMs





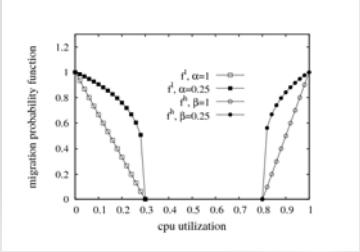
VM assignment procedure

1. The manager sends an invitation to a subset of servers
2. Each server evaluates the assignment probability function based on the local CPU utilization
3. The server performs a Bernoulli trial to decide whether or not to be available: if so, the server sends a positive ack to the manager
4. The data center manager randomly selects the server that will execute the VM



VM migration procedure

1. A server checks if its load is in the range between a low and a high threshold
2. If the utilization is too low, the server should try to get rid of the running VMs. If the utilization is too high, an overload event may occur in a near future
3. In these two cases, the server performs a Bernoulli trial based on the migration probability function
4. If the trial is positive, a migration is initiated



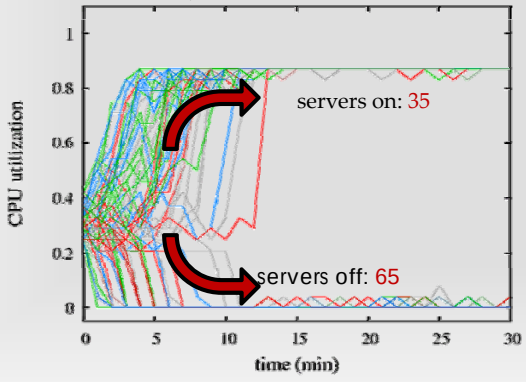
main features of eco4cloud

1. No complex algorithm: decisions are based on local information
2. Scalable behavior, thanks to the probabilistic and self-organizing approach
3. Migrations are gradual and asynchronous
4. Overload events are prevented with timely migrations
5. Same software for all virtualization environments: VMWare, HyperV, KVM…

consolidation with eco4cloud

- Servers are not allowed to stay in a low utilization range
- They either get hibernated or highly utilized

Experiment with 100 servers



With eco4cloud:
35 servers take all the load
65 servers are hibernated

Carlo Mastroianni
Works '12 at Supercomputing 2012, Salt Lake City, November 2012