

II.1. Client, serveur, service, requête, réponse

L'environnement client/serveur désigne une communication à travers un réseau entre deux parties applicatives (logicielles) : typiquement, d'un côté le client envoie des requêtes vers le serveur, de l'autre côté, le serveur attend les requêtes et y répond. Un serveur peut généralement répondre à un grand nombre de clients.

L'environnement client/serveur concerne généralement des machines hétérogènes du point de vue matériel (processeur, câblage ...), système d'exploitation (Unix, Windows ...) et éventuellement applicatif (langages C, Java ...) d'où la nécessité d'établir des protocoles de communication.

On parle souvent de serveur pour désigner la machine hébergeant le logiciel serveur. Celle-ci est souvent dotée de capacités supérieures en terme de puissance de calcul, de volume de stockage, d'entrées/sorties, de connexions réseaux.

De même, on parle aussi de clients pour désigner les ordinateurs hébergeant les logiciels clients. Il s'agit souvent d'ordinateurs personnels ou d'appareils individuels (smartphones, assistants personnels, tablettes).

Le serveur (machine) doit être tolérant aux pannes : redondance d'alimentation, redondance des services et données (sur plusieurs supports).

Précision sur le terme client/serveur

Certains services réseaux sont beaucoup utilisés et on a pris l'habitude de désigner par serveur une machine qui héberge des services : serveur Web (aussi appelé serveur HTTP), serveur de courrier (serveur SMTP) ...

Mais

Le concept de serveur désigne en fait un service de la même façon que le client désigne le programme sollicitant ce service.

Ainsi

Une même machine peut ainsi héberger plusieurs services (HTTP, SMTP, FTP etc.)

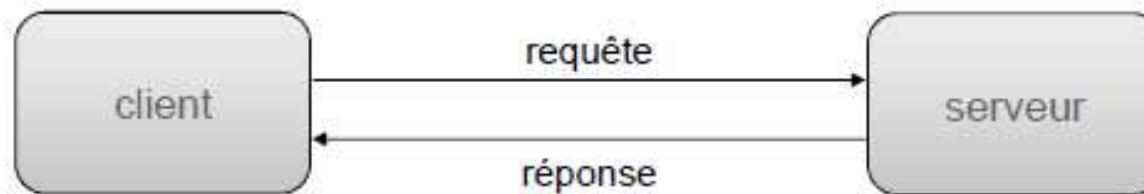
Une même machine peut être tour à tour client et serveur, ce qui est par exemple le cas lorsque :

- un serveur SMTP reçoit un mail et le transfère vers un autre serveur SMTP
- un Peer reçoit une demande de la part d'un Peer entrant et la transmet vers d'autres Peers (sortants).

Le plus souvent, le client initie le dialogue ; il demande l'exécution d'un service (mode Pull)

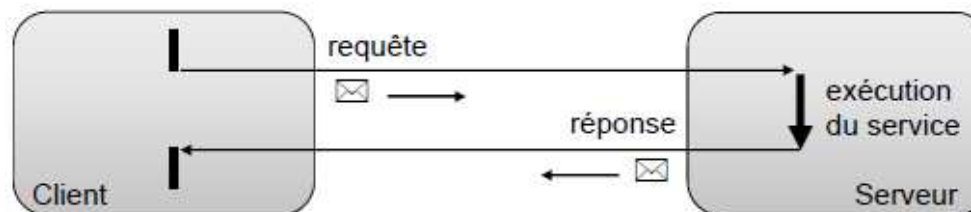
Le serveur est à l'écoute d'une requête cliente; il y répond en réalisant le service (service rendu = traitement effectué par le serveur)

Client et serveur sont en général localisés sur deux machines distinctes mais il peut s'agir de la même machine (localhost)



Cas typique : Appel de procédure distante (RPC)

- Requête : spécification du service requis, paramètres d'appel
- Réponse : résultats, indicateurs éventuels d'exécution ou d'erreurs



Serveur en mode Pull ou en mode Push

Classiquement, le serveur est en mode Pull : le client initie le dialogue, il tire (Pull) vers lui l'information

Dans le mode Push, le dialogue est lancé par le serveur

- Par exemple, le client peut être conçu dès le départ pour devoir accepter les mises à jour (ce qui impose une autorisation préalable de la part du client). Autre exemple, le client s'inscrit à un groupe/topic et se voit ensuite transférer chaque nouvelle information émise sur ce groupe (modèle publication/abonnement).
- Le serveur peut initier un dialogue (mises à jour de programme) ou bien être un intermédiaire dans une communication plus globale (visioconférence, transfert de courrier)
- Communication push du serveur vers des clients (tchat, visioconférence ...) ou vers d'autres serveurs (transfert de courrier électronique ...)
- Faux Push : beaucoup de clients de messagerie moderne simulent le Push en interrogeant fréquemment le serveur POP sur l'arrivée éventuelle de courrier. Par contre, le protocole de relève de courrier IMAP supporte nativement le Push.

Architecture client/serveur : avantages et inconvénients

Avantages

L'ensemble des services et des données étant centralisées sur une même machine, cela facilite l'administration, la mise à jour des programmes et des données, les contrôles de sécurité, etc.

Cette centralisation peut avoir d'autres avantages : par exemple favoriser l'indexation et la recherche d'informations sur le Web (moteurs de recherches)

Inconvénients

Si le serveur n'est plus disponible, les clients ne fonctionnent plus (intérêt de la redondance de services, de données ...)

Le serveur supporte toute la charge de travail ; il peut saturer si trop de clients se connectent en même temps

Les clients ne communiquent pas entre eux (asymétrie de l'information au profit du serveur)

Remarque : le Peer To Peer est un cas particulier de client/serveur puisque chaque machine joue à la fois le rôle de client et de serveur (décentralisation des données, charge de travail répartie ...)

II.2. Serveurs Web, de fichiers, de messagerie ...

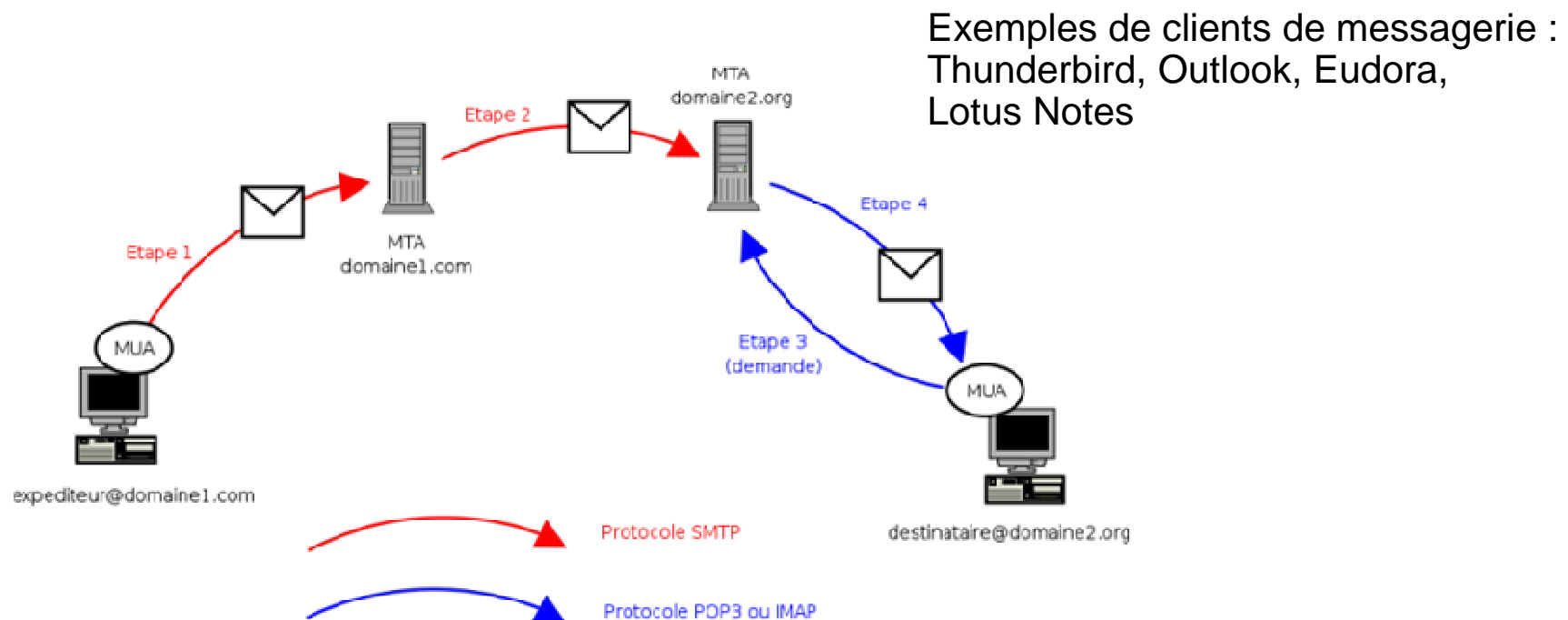
Serveur Web : Fournit des pages Web à des navigateurs clients (Internet Explorer, Firefox ...) qui en font la demande sur le réseau grâce au protocole HTTP. La machine doit généralement disposer de capacités d'E/S élevées.
Exemple : Apache, Internet Information Services (IIS)

Serveur de fichiers : Permet de partager des fichiers sur un réseau grâce à un protocole de partage de fichiers, typiquement FTP sur internet. Côté client, les utilisateurs peuvent utiliser des outils tel que Filezilla, CuteFTP ... Sur les réseaux locaux : SMB, NFS ...

Serveur d'impression : Permet de partager une imprimante entre plusieurs ordinateurs à travers le réseau. Les documents en attente sont placés dans des files d'attente sur disque dur (spooling). Il existe de nombreux protocoles (IPP, LPD, CUPS) et pilotes clients associés.

Système de Gestion de Bases de Données : Permet à des utilisateurs d'interagir avec une Base de Données. Exemples de SGBD : Oracle, SQL Server Plusieurs protocoles réseaux (RDA, ODBC, JDBC) et de nombreux pilotes clients (Microsoft *ODBC Driver For SQL Server* ...)

Serveur de messagerie : Communique avec un client de messagerie (MUA : Mail User Agent) ou un autre serveur de messagerie (MTA : Mail Transport Agent) via le protocole SMTP (on parle ainsi de serveur SMTP). Classiquement, le MUA envoie un courrier à son MTA qui le transfère vers le MTA du destinataire. Le MTA du destinataire délivre alors le courrier au serveur de courrier électronique entrant (MDA : Mail Delivery Agent), qui stocke alors le courrier en attendant que l'utilisateur vienne le relever. Selon le protocole utilisé, on parle alors de serveur POP ou de serveur IMAP.



Serveur Usenet (serveur de groupes ou Newsgroup) : abonnement à des groupes (de discussions ou de nouvelles) et partage d'articles au sein de ces groupes. Pour lire ou écrire des articles sur ces forums, les utilisateurs utilisent un "lecteur de nouvelles". Utilisation des protocoles UUCP (UNIX/avant internet) ou NNTP (accessibilité depuis Internet).

Serveur de noms (annuaire des services) : permet de résoudre les correspondances adresses IP \Leftrightarrow nom de domaine. Système de serveurs distribués. Organisation hiérarchique. Utilisation du protocole DNS.

Serveur d'applications : logiciel proposant un environnement d'exécution pour des composants applicatifs (EJB, JSP, Servlets ..., composants ASP.NET ...). Exemples : WebSphere/JBoss/Glasfish (J2EE), Microsoft .Net Ils intègrent généralement un serveur HTTP.

Serveurs (protocoles) de type "session distante" (telnet, rlogin, ssh) : ils permettent l'ouverture d'une session depuis une machine distante. PuTTY : client telnet, rlogin, ssh à l'origine pour les systèmes Windows.

Superviseur d'administration réseau : permet la gestion (supervision, diagnostic) d'équipements réseaux (routeurs, serveurs). Utilisation du protocole SNMP

Serveur de temps : permet à des ordinateurs du réseau de synchroniser leur horloge sur un ordinateur ayant une horloge de référence (de précision élevée) ; la communication est basée sur le protocole NTP (Network Time Protocol).

II.3. Architectures client/serveur

Mainframe : ordinateur principal qui contient l'ensemble des traitements et des données. Ancêtre du client serveur "classique". Les clients sont généralement des terminaux passifs (écran + clavier sans unité centrale ni disque dur).

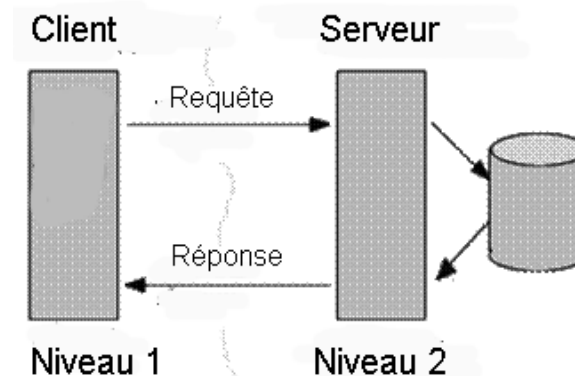
On les trouve dans les très grandes entreprises : banques, compagnies aériennes ...



The IBM System z10 Business Class

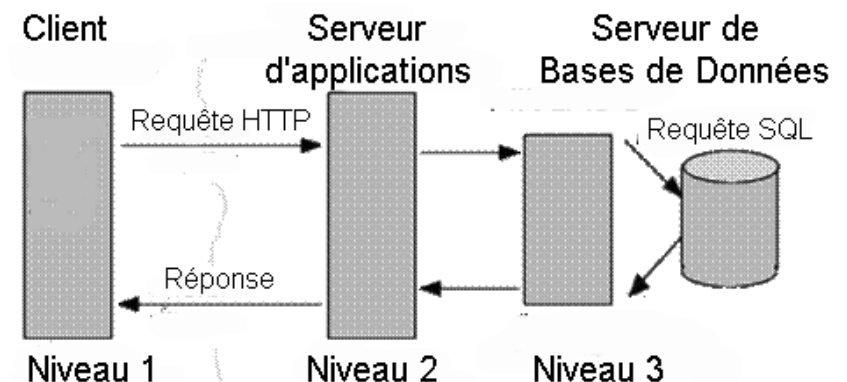
Architecture à 2 niveaux : architecture client/serveur classique ; le client émet des requêtes vers un serveur qui renvoie des réponses ou des ressources.

Exemple : navigateur / serveur Web



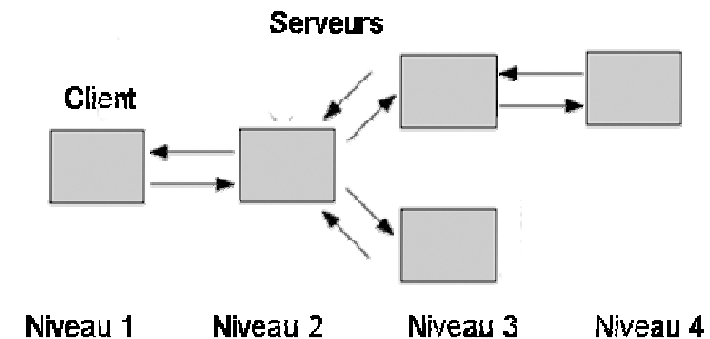
Architecture à 3 niveaux (trois tiers) ; le client est chargé de la présentation (interface graphique) ; le serveur d'application (middleware) est chargé des traitements et fait appel à un serveur de données ; le serveur de données gère les données (stockage, restitution, contrôle).

Exemple : navigateur / serveur Web / SGBD



Architecture à N niveaux : on peut généraliser l'architecture 3-tiers à N niveaux en spécialisant les serveurs dans des tâches de plus en plus précises : un serveur peut utiliser les services d'un ou plusieurs autres serveurs.

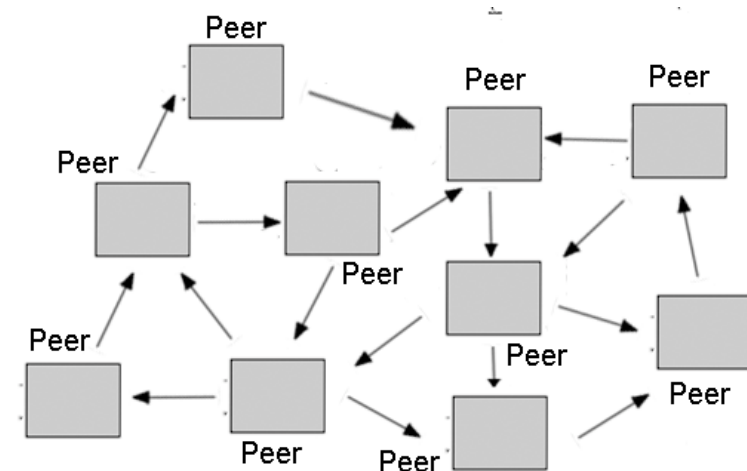
Exemple : le webmail, un client de messagerie qui s'exécute sur un serveur Web



Architecture en Peer to Peer : chaque ordinateur du réseau est susceptible de jouer tour à tour le rôle de client et de serveur.

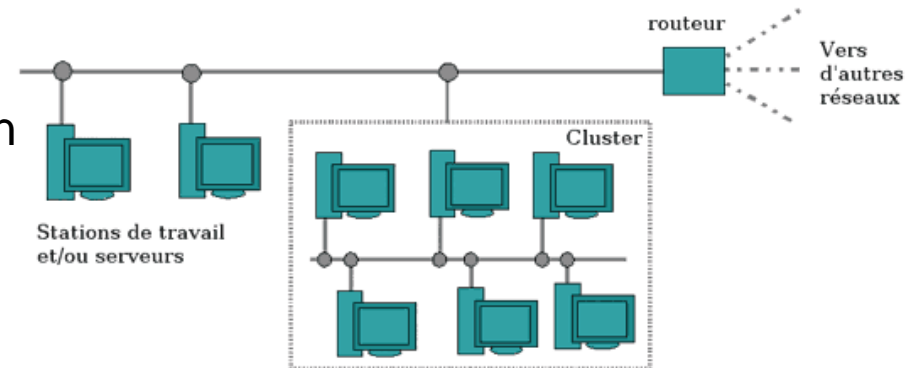
Exemple : les outils d'échange de ressources distribués sur internet (eMule, BitTorrent ...)

Remarque : si le transfert de ressources est toujours distribué, l'administration du réseau lui-même (raccrochement des Peers au réseau, recherche de ressources) peut parfois être confiée à un serveur central (P2P centralisé Vs décentralisé)



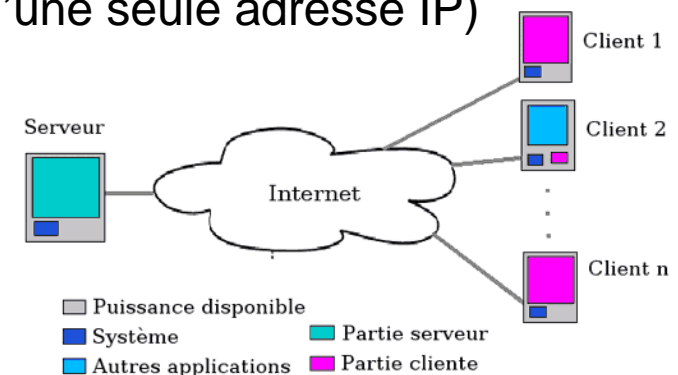
Grappe de serveurs (= cluster = ferme de calcul) : il s'agit de dépasser les limitations physiques d'un seul serveur en utilisant plusieurs serveurs. L'ensemble des serveurs forme un agrégat qui est vu comme une seule entité informatique.

Exemple : serveurs maître/esclaves pour la répartition de charge, la réplication de données, la parallélisation de calculs ...



Grilles de calcul : contrairement aux grappes de serveur, les grilles de calculs concernent des machines hétérogènes (ordinateurs, smartphones ...) et délocalisées sur le réseau. Un cluster peut être vu comme une seule entité physique et fonctionnelle (possibilité d'une seule adresse IP) ce qui n'est pas le cas d'une grille de calcul

Exemple : le projet DataGrid mené par le CERN et soutenu par l'UE permettant aux scientifiques de disposer de la puissance de calcul et des capacités de stockage d'une grille d'ordinateurs délocalisés.



II.4. Répartition de charge : clients lourds, légers, riches

Un client lourd est un client qui doit effectuer en local une partie importante des traitements.

Avantages :

- Décharger le serveur d'autant de traitements sur l'ensemble des clients
- Applications théoriquement moins consommatrices de réseau

Inconvénients :

- Solutions plus coûteuses au niveau du déploiement, de la maintenance, de l'évolution de l'application et de la formation des utilisateurs

Le plus souvent, au lancement de l'application, un appel sur le serveur vérifie qu'il n'existe pas une version plus récente et propose de la télécharger si c'est le cas

Exemple : clients de messagerie courants (Outlook ...)



Client de messagerie Outlook



Interface Java Swing



Skype

Un client léger est un client qui ne fait que formuler les demandes et les présenter à l'utilisateur, les traitements étant entièrement effectués par le serveur.

Avantage :

- L'évolution concerne le serveur uniquement, on évite ainsi N mises à jour sur les clients

Inconvénients :

- On n'exploite pas le parallélisme potentiel client/serveur : le serveur supporte à lui seul toute la charge de travail
- Encombrement réseau généralement plus important
- Interaction homme/machine limitée par nature

Exemples :

- Outils en ligne de commande (terminaux passifs)
- Applications Web (affichage HTML)



Formulaire Web classique : la plupart des traitements sont effectués côté serveur et donnent lieu à un rechargement complet de la page à chaque fois

Un client riche : est généralement utilisé pour évoquer un client Web dont les caractéristiques deviennent similaires aux logiciels traditionnels (interactivité, vitesse d'exécution).

Un client riche cherche à combiner les avantages du client lourd et du client léger.

Les Rich Internet Application sont apparues pour apporter une plus grande richesse dans les IHM des applications Web. Les possibilités des IHM se rapprochent ainsi des IHM traditionnelles.

Exemples : l'avènement de la technologie AJAX avec le Web 2.0.

Concernant les applications Web, on peut parler de client léger pour une application où le serveur retourne des pages 100 % HTML vers le client, de client lourd pour une application effectuant une grande partie des traitements côté client (JavaScript), et de client riche concernant des technologies plus récentes comme AJAX, interactives côtés client mais laissant faire tout le travail au serveur

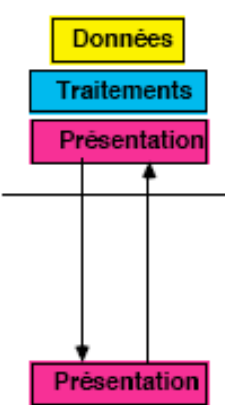
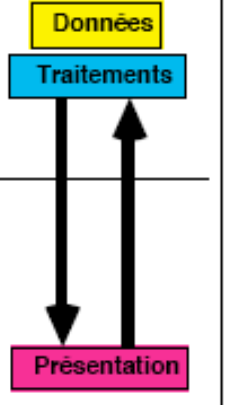
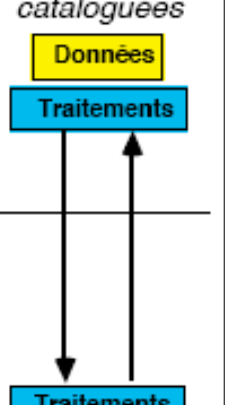
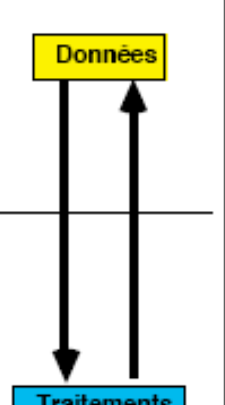
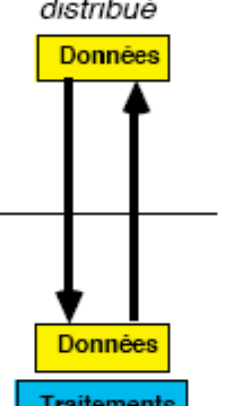
The image shows a web form with several input fields and real-time validation feedback:

- Prénom**: Input field containing "Fabien" with a blue checkmark icon to its right.
- Nom**: Input field with an orange error icon and the message "Votre nom de famille est requis".
- Nom d'utilisateur**: Input field containing "FabienD" with a blue checkmark icon and the message "FabienD est disponible".
- Mot de passe**: Input field containing "****" with an orange error icon and the message "Saisissez au moins 5 caractères".
- Confirmez le mot de passe**: An empty input field.
- Adresse e-mail**: An empty input field.

Formulaire AJAX : vérification de la saisie à la volée

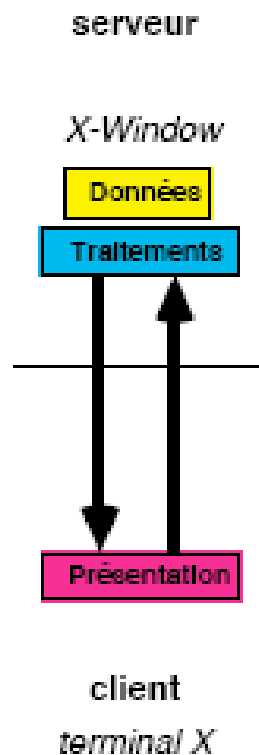
II.5. Le schéma du Gartner Group

Le Gartner Group propose une classification selon la manière de répartir les fonctions de présentation (IHM), de traitement (fonctions, calculs) et de gestion des données (stockage, restitution, contrôle) entre le client et le serveur

<i>Présentation distribuée</i>	Présentation déportée	Traitement distribué	Gestion distante des données	Bases de données distribuées
<i>Revamping</i> 	<i>X-Window</i> 	<i>procédures cataloguées</i> 	<i>R.D.A</i> 	<i>R.D.A distribué</i> 
ne peut être considéré comme C/S	C/S de présentation	C/S de traitement	C/S de données le plus connu et le plus répandu	C/S de données distribuées

Client/serveur de présentation : le client est dédié à la présentation des données et à l'interaction avec l'utilisateur

Exemple : le système graphique X Window (basé sur le protocole X). On sépare la logique d'affichage (client) de sa gestion (serveur). Le serveur X se trouve sur la machine locale (celle de l'utilisateur). Il gère l'écran d'affichage, le clavier, la souris. Les applications clientes (traitement de texte, jeux ...) peuvent être situées à distance. Communication client/serveur via le protocole X.



- Le client émet vers le serveur des requêtes d'affichage (utilisation de la bibliothèque XLib) et reçoit des événements

Requête : par exemple "création d'une fenêtre avec telles dimensions à telle position "

- Le serveur gère des requêtes d'affichage et émet des événements

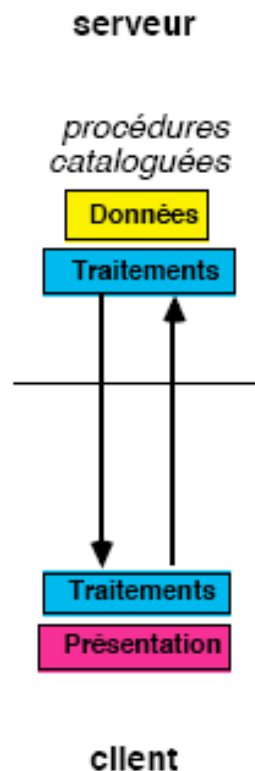
Evènements : clavier, souris, demande de réaffichage (le serveur X communique avec un client particulier, le gestionnaire de fenêtres)

Avantage : séparation nette entre la logique de présentation et l'IHM proprement dite

Inconvénient : trafic réseau important généré par le protocole X

Client/serveur de traitement : les traitements sont distribués entre le client et le serveur

Exemples : appels de procédure distantes RPC/RMI, appels de procédures stockées (PL/SQL) etc.



Procédure PL/SQL stockée sur le serveur

```
CREATE FUNCTION genererIdUser() RETURNS OPAQUE AS
DECLARE
    iduser INTEGER;
BEGIN
    SELECT INTO iduser MAX(id_user) FROM USER;
    IF iduser ISNULL THEN
        iduser := 0;
    END IF;
    NEW.id_user := iduser + 1;
    RETURN NEW;
END;
LANGUAGE 'plpgsql';
```

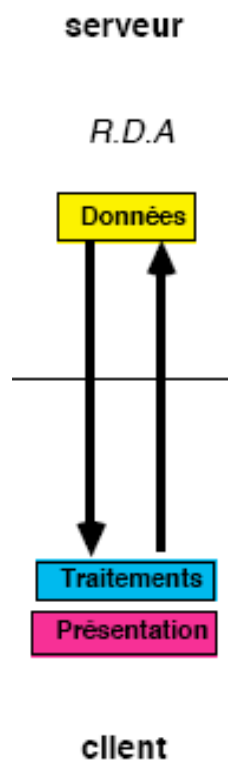
Avantages :

- Permet de bien répartir la charge de travail entre client et serveur
- Généralement, trafic réseau moins important

Inconvénient : il n'offre pas la même modularité que le client/serveur de données (code généralement plus difficile à maintenir)

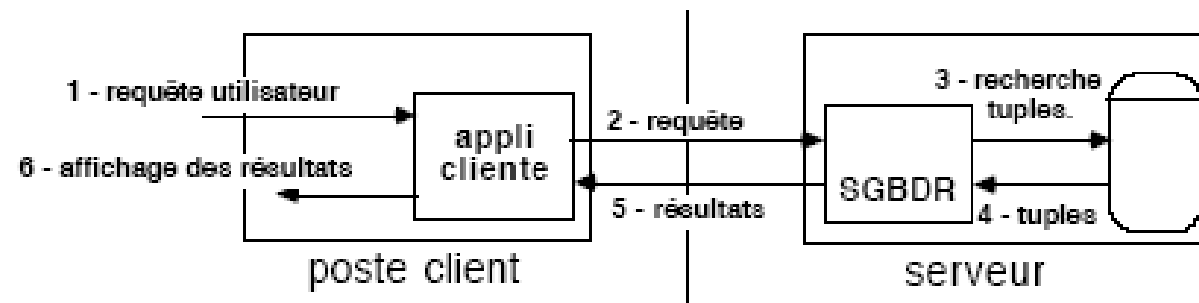
Client/serveur de données : le client contient les traitements tandis que le serveur (SGBD) est dédié à la gestion et au contrôle de l'intégrité des données.

Exemple : accès à un SGBD distant (MySQL) depuis un serveur HTTP (Apache/PHP).



Avantage : indépendance entre traitements et données

Inconvénient : grande charge de travail pour le client



Client/serveur de données distribuées : une partie des données est contenue localement, une autre à distance (décentralisation des données).

Exemples : accès à une base de distante depuis une base de données locale : via une implémentation du protocole RDA ou via l'utilisation de DBLink (objet Oracle permettant de se connecter à une autre base Oracle).

Création d'un DB Link :

```
CREATE PUBLIC DATABASE LINK bdddist  
CONNECT TO SCOTT IDENTIFIED BY 'tiger' USING 'maBase';
```

Dans le fichier 'tnsnames.ora' de la machine locale :

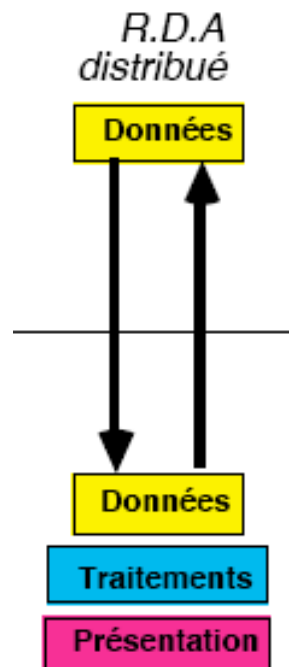
```
maBase= (DESCRIPTION= (ADDRESS_LIST=  
(ADDRESS=(PROTOCOL=TCP)(HOST=135.59.65.54)(PORT=1521) ) )  
(CONNECT_DATA= (SID=distante_db) ) )
```

Utilisation du DBLink :

```
select * from tab1 @bdddist;
```

Avantage : augmentation de la capacité de stockage

Inconvénient : disponibilité et cohérence des données sur N sites difficile à gérer (1 site en panne peut éventuellement rendre le tout indisponible)



I.6. Problématique associée aux serveurs

Problèmes majeurs d'un serveur : la montée en charge, la tolérance aux pannes et la sécurisation des accès

Montée en charge

Deux possibilités pour répartir la charge entre différents ordinateurs d'un groupe :

1. DNS : le serveur DNS peut servir de répartiteur de charge : sur le serveur DNS on associe un nom de domaine à plusieurs adresses IP puis un processus démon nommé 'bind' utilise la méthode du Round Robin pour choisir le destinataire de la requête

2. Load balancing : distribution de la charge de travail et donc des requêtes sur différents serveurs

Répartiteur de charge : C'est l'entité qui contient les algorithmes de répartition. Cela peut être un routeur, un switch, un système d'exploitation ou un logiciel applicatif.

Exemples de solutions logicielles : Linux Virtual Server (LVS), Pen, Balance, Nginx

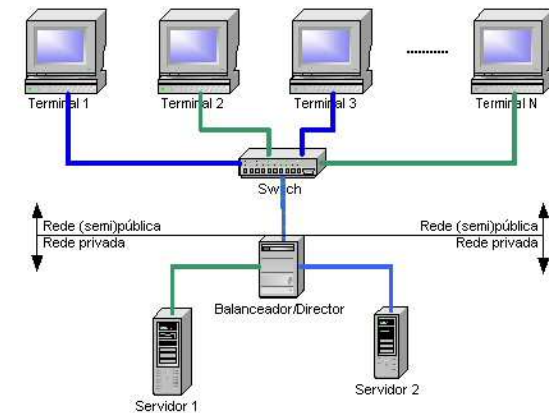
Algorithmes divers : random, round robin (pondéré ou non), moins de connexion (pondéré ou non), table de hachage basée sur l'IP (des serveurs ou des clients), détection de l'indisponibilité des serveurs, conservation de la relation client/serveur lors d'appels ultérieurs ...

Serveur "virtuel" Vs "réel" : le serveur virtuel concentre toutes les requêtes en provenance des clients et les redirige ensuite vers les serveurs réels. Au niveau du serveur virtuel, association de services (http, smtp, dns ou autre) à des serveur réels.

Scalabilité : Capacité d'un produit à s'adapter à une montée en charge en maintenant ses fonctionnalités et ses performances en cas de forte demande.

Selon une étude de 2010, Google est composé de plus d'un million de serveurs dans le monde, pouvant répondre à plus de 34.000 requêtes par seconde. La répartition de charge permet de distribuer les requêtes entre les machines de la ferme.

Calcul parallèle : la répartition de charge peut aussi être utilisée aussi pour décomposer une tâche en plusieurs sous-tâches indépendantes pouvant être exécutés sur différentes machines (par ex : calcul matriciel)



Tolérance aux pannes (haute disponibilité)

Plusieurs notions importantes :

Protection des données : codes détecteurs et correcteurs d'erreurs : bit de parité, code de Hamming, Contrôle de Redondance Cyclique ...

Surveillance applicative et matérielle :

o *Pannes applicatives* :

⇒ ex : Lirectord (associé à LVS) : surveillance applicative de chaque serveur (envoi de requêtes et attente de réponses) et modification en temps réel des règles de redirection.

o *Pannes de machines* :

⇒ ex : Heartbeat : un serveur maître et un serveur esclave. Battements de cœur du maître reçus à intervalle régulier. Si le maître tombe en panne, l'esclave prend le relai (basculement de l'adresse IP).

Réplication de données : Ecriture sur plusieurs supports (disques) simultanément, supports qui peuvent être distants (serveurs maître/esclaves) ou non (même grappe). A ne pas confondre avec la sauvegarde ; la réplication évolue au fur et à mesure de la mise à jour des données, contrairement à la sauvegarde qui est fixée dans le temps à un instant T. Exemples : architecture RAID (disques en grappe), DRBD (= RAID 1 en réseau pour les systèmes Linux).

La technologie RAID

RAID (Redundant Array of Independent Disks) : Groupement redondant de disques Indépendants

RAID permet de constituer une unité de stockage à partir de plusieurs disques durs (grappe de disques)

Il s'agit d'une technologie permettant de répartir l'écriture sur plusieurs disques afin soit :

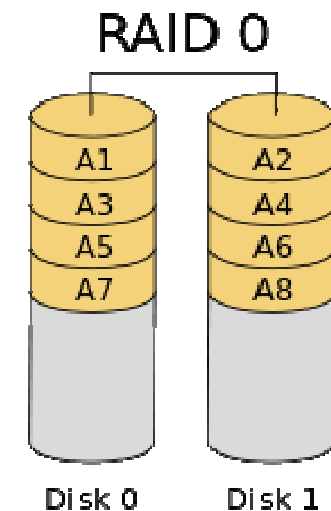
- D'augmenter les performances : découpage d'un ensemble de données et écriture simultanée sur plusieurs disques
- D'augmenter la tolérance aux pannes ceci pouvant être fait de 2 façons :
 - o La réplication de données : mêmes données écrites sur différents disques
 - o La détection et le contrôle d'erreurs (calcul de parité ...)

Il est possible de combiner ces différents avantages en organisant les disques de différentes façon : on parle de niveaux RAID.

Niveaux RAID

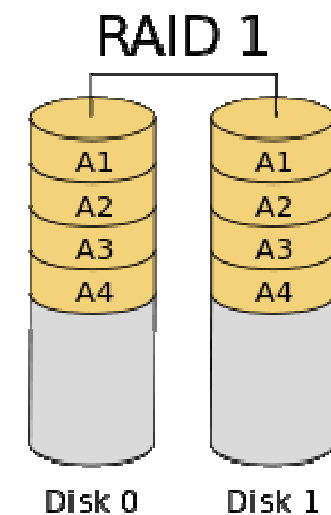
RAID 0

- But : Faire travailler N disque durs en parallèle
- Comment : Ecriture d'un fichier répartie sur N disques
- Avantages/Inconvénients :
 - o Plus rapide : écriture simultanée
 - o Peu fiable : pas de redondance d'informations, ni de mécanismes de correction d'erreurs
- Bonne capacité : Celle du plus petit élément de la grappe x N



RAID 1

- But : Utiliser N disque de manière redondante (avec $N \geq 2$)
- Comment : Les mêmes données sont écrites N fois (miroir)
- Avantages/Inconvénients
 - o Bonne fiabilité : accepte une défaillance de N-1 éléments
 - o Faible capacité : celle du plus petit élément de la grappe
 - o Lenteur due à l'écriture redondante



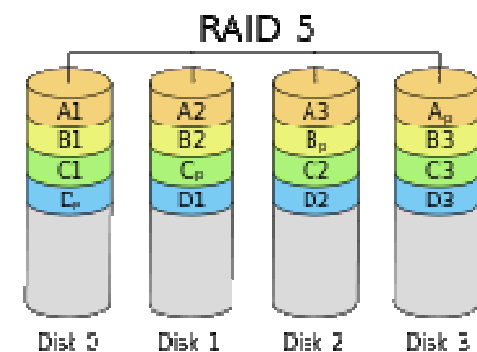
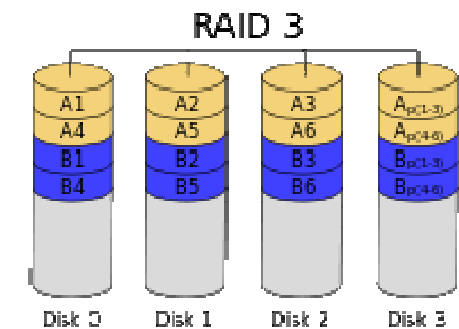
RAID 3 & 4

- But : Parallélisme et correction d'erreurs
- Comment : Utilisation de N-1 disques pour répartir les données (comme RAID 0) et d'1 disque pour la parité
- Avantages/Inconvénients
 - o Fiabilité moyenne
 - Si un disque de données tombe en panne, on utilise les autres disques de données + le disque de parité
 - Si le disque de parité tombe en panne, il peut lui-même être reconstruit à partir des autres disques
 - Si plus d'un disque tombe en panne, les données sont perdues
 - o Capacité, fiabilité et rapidité acceptable

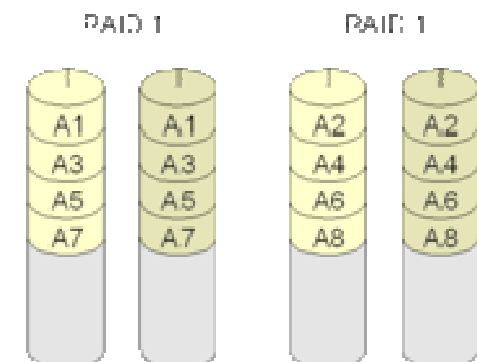
RAID 5 : Utilisation de N disques pour répartir à la fois les données et la parité. C'est une variante de RAID 3 (qui évite le goulot d'étranglement sur un disque dû à l'écriture de la parité), de même fiabilité (pas plus d'un disque en panne)

Combinaison des RAID : RAID 10, 15, 50 ...

Le premier chiffre indique le niveau de RAID des grappes et le deuxième le niveau de RAID global



RAID 10
RAID 0



La sécurisation des accès

1. Firewall

Un Firewall filtre le trafic en autorisant uniquement les accès permis par l'administrateur : filtrage des ports, des protocoles, des adresses IP. Exemple de mise à jour du fichier **/sbin/iptables** (Firewall offert par le noyau Linux) :

Options de la commande **iptables**

-t table (filter par défaut) **-A** ajouter une règle à la fin d'une chaîne (chaîne : INPUT, OUTPUT, FORWARD) **-D** supprimer une règle au sein d'une chaîne **-R** remplacer une règle au sein d'une chaîne **-I** insérer une règle au sein d'une chaîne **-X** effacer une règle au sein d'une chaîne **-F** effacer toutes les règles d'une chaîne **-L** lister toutes les règles d'une chaîne **-P** définir une politique par défaut pour une chaîne **-j** actions à prendre si un paquet répond aux critères de la règle : ACCEPT, REJECT, DROP, LOG ... **-s** adresse IP source à vérifier **-d** adresse IP destination à vérifier **-p** spécifier un protocole (tcp, udp, icmp, all ...) **--sport** port source **--dport** port destination **-i** interface physique d'entrée **-o** interface physique de sortie **--state** état de la connexion (ESTABLISHED, NEW ...)

Exemples

- On accepte toutes les entrées/sorties pour http :

```
iptables -A INPUT -p tcp -dport 80 -j ACCEPT
iptables -A OUTPUT -p tcp -sport 80 -j ACCEPT
```

- On accepte tout le va et vient sur l'interface eth0 :

```
iptables -A INPUT -i eth0 -j ACCEPT
iptables -A OUTPUT -o eth0 -j ACCEPT
```

- On refuse les accès provenant de 152.59.45.36 :

```
iptables -A INPUT -s 152.59.45.36 -j DROP
```

- On accepte tout le va et vient vers le réseau local 192.168.1.0/24 (classe C) :

```
iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
iptables -A OUTPUT -d 192.168.1.0/24 -j ACCEPT
```

2. Outils de détection d'intrusions (IDS) et de prévention d'intrusions (IPS)

Les Firewalls permettent de bannir certains utilisateurs, de bloquer certains ports, mais qu'en est-il des ports ouverts, des attaques en cours ? Il existe des outils de détection et de prévention d'intrusions.

Outils de détection d'intrusion (IDS) :

Rôle d'analyse et de détection de paquets suspects. Trois étapes :

1) Capture : analyseur de paquets *tcpdump* basé sur la librairie *libpcap* (*WinPcap/WinDump* sous Windows).

2) Analyse : Utilisation de bases de signatures (à mettre à jour régulièrement pour les nouvelles attaques).

3) Alerte/Action : écriture dans un système de journalisation, envoi de mail, blocage du trafic, interopérabilité avec d'autres outils de sécurité (écritures de nouvelles règles dans le firewall ...)

Plusieurs types d'IDS selon le périmètre de surveillance :

- Les NIDS (*Network Based Intrusion Detection System*) : au niveau d'un réseau
- Les HIDS (*Host Based Intrusion Detection System*) : au niveau d'un hôte (peut en plus tenir compte des processus, des logs et autres fichiers, des profils utilisateurs) ; ils détectent globalement moins de faux positifs.
- Les IDS hybrides utilisent les NIDS et HIDS

Quelques exemples d'attaques par déni de service :

Le Ping de la mort : envoi de paquets ICMP (ping) avec une taille anormalement élevée. Les paquets sont alors généralement fragmentés mais causent une saturation de la mémoire tampon (pile) lors de leur réassemblage.

Le Packet Fragment : Modifier les numéros de séquence des paquets afin de générer des blancs ou des recouvrements lors du réassemblage. Exemple Teardrop : numéro d'offset du second fragment inférieur à la taille du premier ...

Le Syn flood : Le serveur reçoit un message syn, répond par un syn-ack, mais ne reçoit jamais le dernier ack du client. Accumulation de connexions semi-ouvertes qui finissent par déborder sa mémoire tampon. En pratique, l'attaquant envoie son message syn avec l'adresse IP d'une autre machine (IP spoofing), laquelle, lorsqu'elle recevra le syn-ack du serveur, ne poursuivra pas puisqu'elle n'avait rien demandé (en réalité, comme elle voudra quand même clore la conversation par un rst, l'attaquant, cherchera à faire en sorte qu'elle ne reçoive pas le syn-ack : voir ARP spoofing).

L'UDP flooding : Exploite le fait que le protocole UDP ne fait pas de contrôle de flux et aussi le fait que le trafic UDP est prioritaire sur le trafic TCP. L'exemple le plus connu est la Chargen Denial of Service Attack : il suffit de faire communiquer le service chargen d'une machine avec le service echo d'une autre. Le premier génère des caractères lorsqu'on l'invoque, tandis que le second se contente de réémettre les données qu'il reçoit. Il suffit alors à l'attaquant d'envoyer des paquets UDP sur le port 19 (chargen) à l'une des victimes en usurpant l'adresse IP d'une autre (IP spoofing) et en utilisant le port 7 (echo) comme port source.

Le Smurfing : Cette attaque utilise le protocole ICMP et l'adressage en broadcast : on envoie un ping à toutes les machines). L'astuce est d'émettre ce ping en prenant comme adresse IP source celle d'une autre machine afin que toutes les réponses (echo-request) soit dirigée vers cette machine (la cible de l'attaque).

Quelques exemples d'attaques par usurpation d'identité

L'ARP spoofing : Permet de sniffer le trafic sur le réseau en s'interposant entre une ou des victimes et la passerelle (pour récupérer des mots de passe par ex). L'attaquant inonde le réseau de trames ARP liant l'adresse physique de l'attaquant avec la passerelle. De cette manière, le cache ARP des victimes est corrompu et tout le trafic est redirigé vers le poste de l'attaquant. L'ARP spoofing permet aussi faire du déni de service en associant l'@IP de la passerelle à une adresse MAC inexistante (on coupe ainsi la machine du réseau). Remarque : l'ARP spoofing peut aussi être utilisé dans une attaque syn flood pour éviter que la machine usurpée, en recevant un syn-ack non attendu, ne mette fin à la connexion par un message rst. L'ARP spoofing va permettre d'éviter cela en associant dans le cache ARP de la machine cible, l'adresse IP de la machine usurpée avec une adresse MAC qui n'existe pas. Ainsi la machine usurpée ne recevra jamais le syn-ack.

Le DNS spoofing : de la même façon on peut corrompre le cache DNS d'une machine victime afin de rediriger ses requêtes HTTP vers un ou des sites pirates que l'on contrôle ou de bloquer son accès à certains sites Web (déni de service).

Outils de prévention d'intrusion (IPS)

Ce sont des IDS actifs. Ils essayent de détecter des comportements suspects avant qu'une attaque n'ait lieu. Par exemple, le nombre de sessions à destination d'un port donné dépasse un seuil dans un intervalle de temps donné. Autre exemple, se baser sur les habitudes des utilisateurs (fonctionnement par apprentissage).

A la différence des IDS l'étape d'analyse est plus basée sur des flux (fichiers de logs) que sur des paquets individuels.

Exemples d'IPS :

- Portsentry : utilitaire qui analyse et bloque en temps réel les scans de ports (à la recherche de ports ouverts). L'attaquant est ajouté à /etc/hosts/deny (et/ou iptables).
- Fail2Ban : se base sur les logs de certains services (ssh, ftp ...) de la machine pour repérer les erreurs de mots de passe répétés dans un laps de temps donné. S'il en trouve, il bannira l'adresse IP de l'attaquant via iptables.

Exemple d'IDS :

- Snort : permet de tester les valeurs des différents champs d'en-tête des paquets IP, ICMP TCP, de rechercher du contenu dans la charge d'un paquet ...

Snort

NIDS (pouvant aussi jouer le rôle d'IPS) open source.

Analyse les paquets à partir de règles écrites dans des fichiers **.rules**.

Possibilité d'inclure des .rules les uns dans les autres (**include:...**). Possibilité d'utiliser des variables (**var NOM_VAR / \$NOM_VAR**), possibilité donnée aux utilisateurs d'ajouter des plugins modulaires (les préprocesseurs).

Règle Snort : **entête** + **options**

Exemple : **alert tcp 192.168.1.0/24 80 -> any any (flow:to_server; msg:"usurp server");**

Action de la règle : **alert** (alerte selon la méthode sélectionnée + journalisation), **log** (journalisation seule) ...

Protocole : **tcp**, **udp**, **icmp** ...

Adresses IP : **any** (n'importe laquelle), une adresse IP unique (ex : **192.168.1.2**), un réseau local (ex : **192.168.1.0/24**). Remarque : réseau de classe A=**/8**, B=**/16**, C=**/24**

Numéros de port : unique (ex : **80**) ou dans une intervalle (ex : **1:80**) avec éventuellement une borne min (**500:**) ou max (**:6000**). Possibilité d'exclusion (ex : **!6000:6010**)

Sens dans lequel la règle d'applique : **->** (unidirectionnel) ou **<>** (bidirectionnel).

Options

msg : message à afficher lors d'une journalisation ou une alerte

ttl : teste le champ Time To Live (TTL) de l'en-tête IP

tos : teste le champ Type Of Service (TOS) de l'en-tête IP

id : teste le champ numéro (ID) de fragment de l'en-tête IP

ipopts : teste des codes du champ Options de l'en-tête IP : **rr** (enregistrement d'une route) ...

fragbits : teste les indicateurs de fragmentation du champ Drapeaux de l'en-tête IP : bit **D**

(paquet non fragmentable si 1), **M** (fragment non terminal si 1, dernier fragment si 0) ...

utilisable avec les symboles **+** (tous les bits spécifiés sont positionnés), ***** (au moins un) et **!**

(aucun des bits spécifié n'est positionné).

...

itype : teste le champ Type de l'en-tête ICMP

icode : teste le champ Code ICMP de l'en-tête ICMP

icmp_id : teste le champ ICMP ECHO ID

icmp_seq : teste le numéro de séquence ECHO ICMP

...

flags : teste les bits du champs Code de l'en-tête TCP : bit **F** (FIN), **S** (SYN), **R** (RST), **P** (PSH), **A** (ACK), **U** (URG)

seq : teste le numéro de séquence de l'en-tête TCP

ack : teste le numéro d'acquittement de l'en-tête TCP

...

content : recherche un motif dans la charge utile du paquet (si les données représentent du code binaire, elles doivent être placées entre des |)

content-list : recherche un ensemble de motifs dans la charge utile du paquet

offset : associé à l'option content, fixe le décalage du début de la recherche

depth : associé à l'option content, fixe la profondeur maximale de recherche

nocase : indique que la recherche ne doit pas tenir compte de la casse (min / maj)

...

resp : réponse active (ferme les connexions qui matchent la règle). Utilisable avec les options

rst_snd (message rst à l'envoyeur), **rst_rcv** (rst au receveur), **rst_all** (rst aux deux),

icmp_net (message ICMP_NET_UNREACH à l'envoyeur) ...

react : réponse active (bloque les accès). Utilisable avec les options : **block** (blocage de la connexion), **warn** (envoi d'un message d'avertissement).

threshold : permet de déterminer un seuil de détection. Il utilise lui même plusieurs options :

seconds qui définit la période de temps en secondes, **count** qui définit le nombre d'évènements constituant un seuil dans la période de temps, **type** qui peut prendre les valeurs **limit** / **threshold** / **both** (si le seuil est atteint plusieurs fois dans la période de temps, on prévient 1 fois, plusieurs fois ...), **track** : peut prendre les valeurs

by_src/by_dst (le calcul se fait par adresse IP source/destination).

flow : permet de s'intéresser dans une communication tcp au sens de la communication.

Plusieurs options: **to_server** / **to_client** (il s'agit d'une requête/réponse), **established** /

not_established / **stateless** (la connexion TCP a été établie, non établie, peu importe) ...

3. Connexion sécurisées

Ce sont des protocoles de communication sécurisés qui imposent l'échange de clés de chiffrement en début de connexion. Chaque envoi est ensuite chiffré et authentifié.

Codes à clé secrète (ou symétrique)

Méthode : Une même clé secrète est partagée par l'émetteur et le destinataire du message. Cette clé sert à la fois pour le chiffrement et le déchiffrement (on parle ainsi de code à clé secrète ou à clé symétrique). Le code (méthode de chiffrement/déchiffrement) étant public, sa sûreté repose entièrement sur le secret de la clé

Exemples : le code DES, triple DES, AES

Problèmes :

- Comment s'échanger la clé de façon sécurisée ? La clé peut être interceptée sur le réseau informatique
- La multiplicité des clés : pour un groupe de n personnes il faut distribuer $n \times (n-1) / 2$ clés

Solution : les codes à clé publique

Codes à clé publique (ou asymétrique)

Principe : Dans la cryptographie à clé publique, on utilise 2 clés :

- Une clé publique pour le chiffrement
- Une clé secrète (ou privée) pour le déchiffrement
- Seul le bon couple clé publique/clé secrète permet de lire le message original

Méthode : A doit envoyer un message à B. B génère une clé publique P à partir de sa clé secrète S . A chiffre alors son message avec cette clé publique P : $P(\text{message})$. Seul B peut alors déchiffrer le message puisque sa clé secrète S est complémentaire de la clé publique P : $S(P(\text{message})) = \text{message}$

Exemple : Le code RSA

Inconvénient : les codes à clé publique sont beaucoup plus lents que ceux à clé secrète

Problème : les codes à chiffrement asymétrique permettent de s'assurer que les messages ne seront pas lus par d'autres personnes. Cependant, ils ne permettent pas de s'assurer de l'identité de l'émetteur (quelqu'un peut usurper l'identité d'un autre pour envoyer un message, quelqu'un peut nier avoir envoyé un message)

Solution : la signature numérique

Remarque : un code à clé publique permet de générer une clé publique à partir d'une clé secrète mais pas l'inverse

Signatures numériques

Principe : Les entités communicantes doivent tous les deux posséder un couple clé secrète/clé publique. En reprenant l'exemple précédent, A fabrique P_A à partir de S_A et B fabrique P_B à partir de S_B

Méthode : A calcule $S_A(\text{message})$, puis $P_B(S_A(\text{message}))$ qu'il envoie. A la réception, B calcule $S_B(P_B(S_A(\text{message})))$ et obtient donc $S_A(\text{message})$. Ensuite, il calcule $P_A(S_A(\text{message}))$ et obtient donc message . B peut être sûr que le message provient de A car lui seul connaît S_A . On parle de signature électronique car l'émetteur du message est authentifié.

Remarque : l'échange est maintenant sécurisé dans les deux sens puisque A et B ont tous les deux un couple clé secrète/clé publique.

Inconvénient : deux fois plus long qu'un code à clé publique, lui-même très lent

Problèmes : si les méthodes précédentes permettent de s'assurer que les messages ne seront pas lus par une tierce personne, rien ne permet d'affirmer que les messages ne seront pas altérés lors de la transmission (1) ; d'autre part, la signature électronique n'identifie l'émetteur/le récepteur que de façon relative (2)

Solutions : les fonctions de hachage (1), les certificats numériques (2)

Fonctions de hachage

Principe : Une fonction de hachage calcule le résumé (ou haché) d'un message : $H(\text{message})$. Ce calcul doit être à sens unique. Partant d'un message, il doit être facile d'obtenir son résumé, mais partant d'un résumé, il doit être impossible (ou très difficile) d'obtenir le message original. En calculant le résumé d'un message à l'arrivée et en le comparant au résumé fourni par l'émetteur, on peut s'assurer de l'intégrité du message. En effet, si une personne mal intentionnée modifie le message pendant son transfert, son haché, calculé à l'arrivée sera différent de celui fourni par l'émetteur. De cette façon, le destinataire s'en rendra compte.

Signature numérique avec fonction de hachage :

A calcule d'un côté $H(\text{message})$ puis $S_A(H(\text{message}))$, et d'un autre côté $P_B(\text{message})$ et envoie le tout. A la réception, B calcule d'un côté $P_A(S_A(H(\text{message})))$ et obtient donc $H(\text{message})$ et d'un autre $S_B(P_B(\text{message}))$ et obtient donc message dont il calcule lui même le résumé, $H(\text{message})$. Il peut enfin comparer ce résumé avec celui envoyé par A.

Exemple de fonction de hachage : SHA-1, MD5

Signature numérique avec fonction de hachage : PGP

Certificats numériques

Problématique : Dans tous les exemples de cryptographie à clé publique que nous avons vus, le destinataire est d'une certaine façon authentifié : en utilisant sa clé publique pour crypter un message, on est assuré qu'en possédant l'unique clé privée complémentaire, lui seul pourra lire le message. Cependant cette authentification n'apporte pas une grande sécurité. En effet, un escroc peut se faire passer pour quelqu'un d'autre en émettant une fausse clé publique, récupérer un paiement puis déménager. Pour répondre à ce genre de problèmes, on utilise des certificats numériques.

Méthode : A veut certifier que sa clé publique lui appartient. Il envoie donc cette clé à un organisme de certification, ainsi que différentes informations le concernant (nom, prénom, métier, e-mail, etc.). Cet organisme vérifie alors la validité de ces informations, puis crée un certificat en y ajoutant son propre nom, une date limite de validité. Enfin l'organisme calcule un résumé du certificat, $h(\text{certificat})$, puis le signe avec sa clé privée : $S_o(h(\text{certificat}))$. Lorsque B veut s'assurer de l'identité de A, il récupère le résumé du certificat fourni par l'organisme (en utilisant la clé publique de l'organisme) : $P_o(S_o(\text{certificat})) = h(\text{certificat})$ puis le compare au haché qu'il a calculé de son côté. Il s'assure ainsi que la clé publique de A contenue dans le certificat est donc sûre. B peut donc l'utiliser pour envoyer et recevoir des messages vers/depuis A

Clés de session

Intérêt : Le chiffrement asymétrique permet de s'affranchir du problème d'échange de clé. Cependant, il est beaucoup plus lent que son homologue symétrique. La notion de clé de session est un compromis entre ces deux modes de chiffrement.

Principe : Elle consiste d'abord à échanger une clé secrète grâce à un chiffrement asymétrique, puis à utiliser ensuite cette clé secrète pour communiquer grâce à un chiffrement symétrique.

Méthode : B émet à partir d'une clé secrète S_B la clé publique P_B . A calcule $P_B(S)$ c'est-à-dire qu'il utilise la clé publique de B pour envoyer sa propre clé secrète. A la réception, B calcule $S_B(P_B(S))$ et obtient donc S. A ce stade, B et A connaissent la même clé secrète, S. Il leur est alors possible de communiquer grâce à un chiffrement symétrique.

Les protocoles SSH, SSL, TLS

SSH, SSL et son successeur TLS sont des protocoles de sécurisation des échanges sur Internet.

SSH a donné son nom au programme qui permet le lancement de commandes sur une machine distante (comme telnet / rlogin ...) de manière sécurisé. SSH utilise le principe des clés de session : le client utilise la clé publique du serveur de commandes pour lui envoyer sa clé secrète. Le serveur la conservera pendant toute la session afin de permettre des échanges par chiffrement symétrique.

TLS assure plusieurs choses :

- Confidentialité des données échangées (clés de sessions)
- Intégrité des données échangées (fonction de hachage)
- Authentification du serveur (signature numérique)
- Authentification forte optionnellement (certificats numériques)

On peut les utiliser avec d'autres protocoles :

- SFTP = FTP + SSH
- FTPS = FTP + TLS
- HTTPS = HTTP + TLS

etc.

II.7. Modes d'interaction client/serveur

Mode itératif ou concurrent : requêtes traitées en séquence ou en même temps

- Mode itératif : Le serveur ne traite qu'avec un seul client à la fois. Le processus de traitement sur le serveur peut donc à priori être unique et répété en séquence.
- Mode concurrent : Le serveur peut traiter avec N clients en même temps. Il peut donc y avoir N processus de traitement à un instant T sur le serveur (en plus du processus d'écoute, qui lui tourne en permanence). Principe du veilleur-exécutants. Le veilleur (processus d'écoute) attend les nouvelles demandes (il est donc toujours actif). Pour chacune, il déclenche un exécutant (processus de traitement).
 - Parallélisme réel (multiprocesseurs) : Les exécutants ne sont pas interrompus (chacun dispose de son propre processeur et de son propre espace mémoire).
 - Pseudo parallélisme (monoprocesseur) : Les exécutants commutent au niveau du processeur pour être exécutés à tour de rôle (notions de sauvegarde et de restauration de contexte). Soit ils partagent un espace mémoire commun (processus légers ou threads), soit chacun dispose de son propre espace mémoire (fork)

Espace mémoire partagé : synchronisation de l'accès aux ressources

Verrous : Les deux seules opérations sur un verrou sont 'verrouiller' et 'déverrouiller'. Un processus ne peut pas accéder à la ressource tant qu'il n'a pas obtenu le verrou sur celle-ci. Dès qu'un processus a fini d'utiliser la ressource il libère le verrou. On dit que la ressource est en exclusion mutuelle.

Attention aux interblocages : exemple : le processus P1 obtient un verrou sur la ressource R1 et le processus P2 obtient un verrou sur la ressource R2. Mais par la suite, P1 a besoin d'accéder à R2 et P2 à R1 ; blocage à prévenir ou à détecter.

Sémaphores : Mécanisme un peu plus élaboré que le verrou car proposant un certain nombre de jetons. Le nombre de jetons correspond au nombre de processus pouvant partager la même ressource.

Producteur(s) / Consommateur(s) : Mécanisme plus élaboré que le sémaphore puisqu'on veut obliger les processus à respecter une certaine règle du jeu. Un producteur ne peut pas écrire dans le tampon tant qu'il n'est pas vide et un consommateur ne peut pas lire dans le tampon tant qu'il n'est pas plein. On utilise alors en plus des booléens.

Moniteurs : Les processus se synchronisent directement entre eux. A un moment donné, un seul processus n'est actif dans le moniteur. Lorsque le processus actif ne peut progresser dans son travail, il se bloque et libère l'accès au moniteur : fonction wait(). Il peut alors réveiller un processus en attente du moniteur qui était

Communication en Point à Point ou multipoints

- Point à point : Il y a un émetteur et un récepteur
 - o Unicast : le récepteur est identifié à l'avance
 - o Anycast : on choisit un récepteur parmi plusieurs possibles (le plus proche, le plus disponible ...)
- Multidiffusion (multipoints) : il y a un émetteur et N récepteurs. Deux possibilités :
 - o Broadcast : L'émetteur connaît ses récepteurs (un réseau local par exemple)
 - o Multicast : l'émetteur ne connaît pas ses récepteurs : mode publication/abonnement (les publications sont émises dans un groupe, n'importe qui peut s'abonner ou se désabonner au groupe) sans que l'émetteur ne le sache . Les tranches d'adresse IP allant de 224.0.0.0 à 239.255.255.255 sont réservées pour le multicast

Mode connecté ou mode non connecté : Nécessité d'établir une connexion entre le client et le serveur ou pas. Une connexion peut comprendre une étape d'identification (login & mot de passe), la négociation de paramètres pour l'échange, l'envoi d'accusés de réception, la possibilité de faire du transactionnel

...

Communication éphémère ou persistante

- Communication éphémère :

Le message n'est conservé que tant que le client et le serveur sont en exécution ; si il y a un problème, le message est perdu. C'est le cas d'une communication téléphonique, de la messagerie instantanée... Les routeurs fonctionnent aussi sur ce principe.

=> Exemples : sockets, RPC

- Communication persistante :

Si l'émetteur s'arrête une fois son message envoyé ou si le destinataire n'est pas fonctionnel, le système de communication stocke sur disque le message tant qu'il le faut. C'est le cas d'une communication par mail.

=> Exemples : JMS (MQSeries, MSMQ), CorbaMessaging ...

Remarque : même si la couche socket/RPC n'est initialement pas prévue pour ça, rien n'empêche un programmeur de gérer lui-même la persistance des données dans son programme (écriture des échanges sur le disque dur). A contrario, on peut très bien programmer avec JMS et décider de ne pas stocker les messages (messages non persistants/mode push)

Mode avec état ou sans état : Existence ou non d'un jeu de données associées à chaque relation client/serveur.

- Mode sans état (Stateless) :

Les appels peuvent modifier des données globales (ex : un compteur de visites) au niveau du serveur mais ils ne disposent pas de leur propre jeu de données. Les appels successifs s'exécutent donc sans liens entre eux, toutes les caractéristiques sont passées au moment de l'appel. Exemple : HTTP.

- Mode avec état (Statefull) :

Chaque appel est associé à un jeu de données spécifique, lequel représente un état de la relation client/serveur (historique des pages activées, panier de commande, position courante dans l'arborescence du serveur ...). Un appel peut donc dépendre des appels antérieurs. Exemple : FTP, Telnet, SMTP

Remarques :

- On peut faire de la gestion d'état de façon persistante (hors communication). Exemple : dans une application de type FTP garder un pointeur sur le dernier fichier ouvert par l'utilisateur lors de sa dernière connexion.

- HTTP est un protocole sans état mais rien n'empêche de faire de la gestion d'état "au dessus", c'est-à-dire dans l'application elle-même. Exemple : utilisation de l'objet HttpSession des servlets (Java)

- Les données sont souvent stockées côté serveur mais rien n'empêche de les conserver côté client. => Exemples : cookies (applications Web)

Communication synchrone ou asynchrone

- Communication synchrone (bloquante) :

Les deux entités communicantes doivent être présentes en même temps. Elles émettent à tour de rôle, l'activité de l'une étant suspendue pour attendre la réponse de l'autre. C'est le principe d'une conversation par talkie-walkie où l'un parle puis l'autre répond.

=> Exemples : sockets, RPC

- Communication asynchrone (non bloquante) :

Les entités ne sont pas forcément actives en même temps et un message peut donc être traité ultérieurement. C'est le principe du mail. Notion étroitement associée à celle de persistance. L'avantage est que l'émetteur n'attend pas la réponse du destinataire et peut continuer à faire autre chose. L'inconvénient est qu'il est plus compliqué de savoir si un message a bien été reçu et comment se place cette réception par rapport aux actions faites par l'émetteur après l'envoi.

=> Exemples : JMS (WebSphere MQ, Joram ...), MSMQ, CorbaMessaging ...

Communication isochrone :

Les temps de réponse du serveur sont bornés (temps min et max)

Remarque : Faire de l'asynchrone avec les RPC :

1) en utilisant deux processus (pseudo-) parallèles sur le client, l'un pour les appels distants et l'autre pour les traitements locaux

2) En permettant au serveur de répondre en

exécutant à son tour une procédure sur le client

Questions de cours

- 1) Classer les RAID 0,1 et 3 en terme de rapidité, capacité et fiabilité.
- 2) Concernant les connexions sécurisées, indiquer si les affirmations suivantes sont vraies ou fausses :
 - a) Un code à clé secrète est plus rapide qu'un code à clé publique.
 - b) Un code à clé publique ne peut être utilisé que pour identifier le récepteur d'un message (c'est-à-dire s'assurer que seul celui à qui est destiné le message pourra le lire).
 - c) Lors de l'envoi d'un message M de A vers B, si A veut s'assurer que seul B pourra lire son message, il va devoir le chiffrer avec P_B .
 - d) Lors de l'envoi d'un message M de A vers B, si A veut prouver à B que c'est bien lui qui envoie le message, il va devoir le chiffrer avec P_A .
 - e) Une clé de session est une clé secrète récupérée grâce à un code à clé publique.
- 3) Dans le développement d'une application répartie côté serveur, dans quel(s) cas pourrait t-on avoir besoin de synchroniser les accès aux ressources (verrous ...)
- 4) Pourquoi les notions d'asynchronisme et de persistance sont-elles liées ?
- 5) En quoi les cookies permettent-ils à HTTP de faire de la gestion d'état ?