La gestion des réseaux IP basée sur les technologies Web et le modèle *push*

GRES'99, Montréal, Québec, Canada 7-10 juin 1999

Jean-Philippe Martin-Flatin
École Polytechnique Fédérale de Lausanne
Institut pour les Communications informatiques et leurs Applications
ICA
martin-flatin@epfl.ch
http://icawww.epfl.ch/~jpmf/

Plan

- Problèmes liés à la gestion de réseau basée sur SNMP
- Solution proposée :
  - gestion de réseau basée sur les technologies Web
  - modèle *push* pour la gestion régulière
  - modèle *pull* pour la gestion *ad hoc*
- Vue d'ensemble du prototype JAMAP
- Conclusion

Gestion des réseaux IP aujourd'hui

- Cadres de gestion SNMP (v1, v2c, v3)
  - paradigme gestionnaire-agent
  - *polling* pour le *monitoring* et la collecte de données (modèle *pull*)
  - notifications (modèle *push*)
  - agents simples et "idiots": le gestionnaire fait tout le travail
- Protocoles de communication SNMP (v1, v2c, v3)
- Plateformes de Gestion de Réseau (PGR) : HP OpenView...

Tâches obligatoires :	**Tâches facultatives :**
- *monitoring* - collecte de données --> rapports - traitement des notifications	- gestion de configuration - gestion d'inventaire - gestion des contrôles d'accès - facturation...

- *Add-ons* = extensions propriétaires (par ex. CiscoWorks)

Problèmes liés aux PGR (1/2)

- Pour les clients :
  - les PGR coûtent trop cher (matériel et logiciel) :
    - ➠ matériel dédié à la gestion de réseaux
  - le support de BD relationnelles tierce-parties est trop limité :
    - ➠ le client dépend d'accords bilatéraux entres vendeurs de PGR et de BD rel.
  - coût de migration Unix --> Windows trop élevé :
    - ➠ expertise Unix nécessaire pour la maintenance des PGR existants
    - ➠ investissement lié à un processeur et à un système d'exploitation
- Pour les fournisseurs de matériel réseau :
  - coût de développement des *add-ons* propriétaires trop élevé :
    - ➠ nombreux PGR --> nombreuses API
    - ➠ nombreux systèmes d'exploitation
    - ➠ nombreux *add-ons*

Problèmes liés aux PGR (2/2)

- Pour les clients comme pour les fournisseurs :
  - temps de mise sur marché des *add-ons* trop long :
    - ➠ si grosse part de marché : plusieurs mois après la mise en vente du matériel
    - ➠ si faible part de marché (par ex. start-up) : jamais
      - recours à un PGR dédié
      - la gestion de réseau n'est plus intégrée
  - gestion de versions multiples d'une MIB :
    - ➠ mise à jour d'une MIB dans le réseau --> pas la même version côté agent et côté gestionnaire (*add-on*) :
      - soit on met le PGR à jour manuellement, après chaque *upgrade* (car pas de protocole de découverte automatique de MIB)
      - soit on n'utilise pas les nouvelles fonctionnalités de la MIB tant que tous les équipements du réseau n'ont pas été mis à jour

Problèmes liés à SNMP (1/3)

- L'expertise SNMP est rare et chère (notamment SNMPv3)
- Certains choix conceptuels initiaux (SNMPv1), inchangés pour des raisons de compatibilité, ont un impact négatif sur la *scalability*, l'*overhead* réseau et/ou la latence :
  - encodage BER inefficace
  - mécanisme bizarre pour transférer un tableau de données SNMP (pas de `get-table`)
  - la description d'un OID prend bien plus de place que sa valeur
  - pas de compression des données de gestion

Problèmes liés à SNMP (2/3)

- La sémantique offerte est de bas niveau :
  - informations de type instrumentation
  - pas d'API standard de haut niveau
  - les applications de gestion des réseaux IP sont :
    - redéveloppées à partir de zéro pour chaque client
    - dépendantes de l'API d'un PGR donné, et non d'une technologie standard
- Securité :
  - SNMPv1 et SNMPv2c : aucune
  - SNMPv3 : possible, mais pas encore utilisée
  - gestion de bureaux distants via RPV : matériel d'encryptage coûteux
  - coupe-feu : relais UDP (config. non triviale pour une PME)

Problèmes liés à SNMP (3/3)

- Protocole de transport non fiable :
  - des notifications importantes (non acquittées) sont perdues pour des raisons futiles (ex.: débordement de mémoire tampon dans un routeur)
  - *inform* de SNMPv3 (notification acquittée) : pas encore utilisé
  - conséquence : les données de gestion critiques nécessitent que les retransmissions soient effectuées au niveau de l'application
- Distribution :
  - entre gestionnaires : aucune (M2M MIB obsolète)
  - entre gestionn. et agent (code mobile) : Script MIB pas encore utilisée
- Évolution ralentie par le poids des systèmes existants :
  - "mieux vaut remplacer que réparer"

Solution proposée (1/4)

- Garder :
  - les MIB
  - le modèle organisationnel
  - avantage = compatibilité avec l'existant :
    - ➠ possible d'avoir recours à des *proxies* ou à des passerelles pour gérer les équipements déjà déployés

Solution proposée (2/4)

- Changer de cadre de gestion :
  - modèle *pull* --> modèle *push* pour les tâches répétitives
  - transférer une partie du travail du gestionnaire vers les agents
- Changer de protocole de communication :
  - SNMP --> HTTP
  - UDP sans connexion --> connexions persistantes TCP
  - compression de type `gzip`
  - nombre illimité de variables de MIB par cycle de *push*
  - encodage BER --> parties MIME + {chaînes de caractères, XML, objets Java sérialisés...}
  - mécanisme naturel de transfert de tableau de données

Solution proposée (3/4)

- Changer de modèle conceptuel pour le PGR :
  - éclater le gestionnaire en trois entités distinctes :
    - station de gestion (butineur Web)
    - serveur de gestion (servelettes Java)
    - serveur de données (BD rel., BD OO, serveur NFS)
  - réécrire le code du gestionnaire :
    - binaire --> servelettes Java
    - indépendant du processeur
    - indépendant du système d'exploitation
    - indépendant de la BD rel. (JDBC)
  - *add-ons* spécifiques coûteux --> applettes Java universelles pas chères
  - machine dédiée pour le PGR --> machine quelconque
  - quelques BD rel. tierce-parties --> BD rel. quelconque

Solution proposée (4/4)

- Changer de modèle conceptuel pour le PGR (suite) :
  - distribution plus facile à réaliser :
    - ➠ au sein d'un gestionnaire : PGR monolithique --> servelettes distribuées; par ex. :
      - une machine/servelette pour la collecte de données
      - une m/s pour les notifications
      - une m/s pour la corrélation d'évènements
      - une m/s pour la génération *offline* de rapports d'utilisation du réseau
    - ➠ entre gestionnaires : application Java répartie standard
    - ➠ entre gestionnaire et agent : code mobile, objets serialisés

Pourquoi HTTP entre agent et gestionn. ? (1/2)

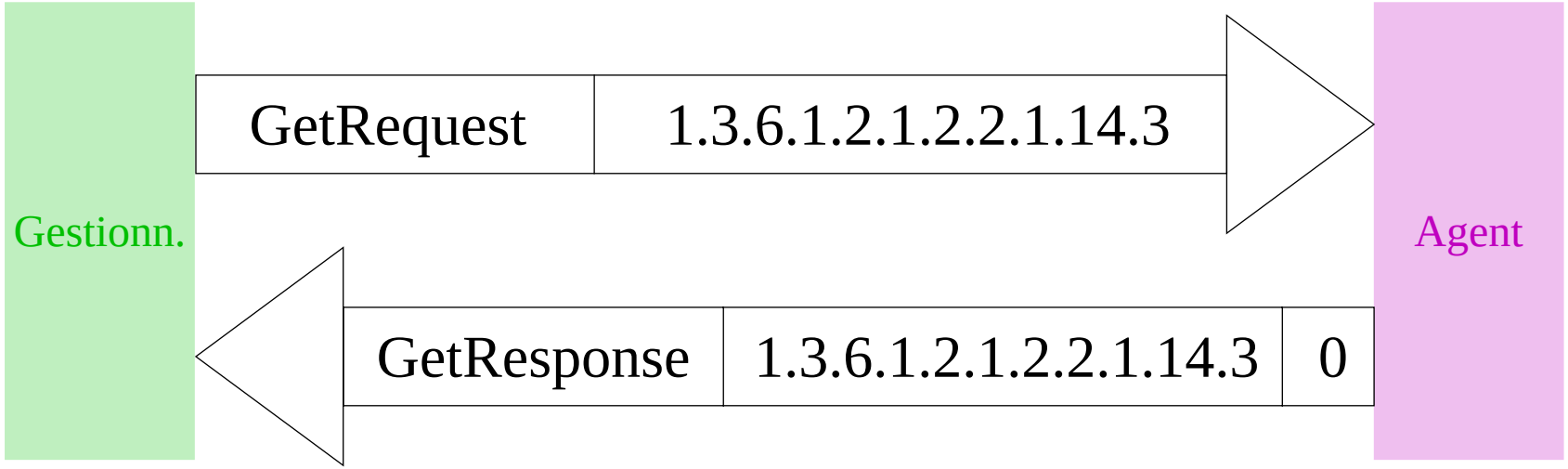
- Quatre techniques pour distribuer une application Java :
  - HTTP
  - socquettes
  - Java RMI
  - Java IDL (CORBA)
- Objets distribués (RMI ou CORBA) en gestion de réseau :
  - télécoms = oui
  - Internet = non
    - licences CORBA trop chères (par ex. coût licence Orbix >> coût petit *hub*)
    - RMI : impact CPU et mémoire trop important sur les agents
    - RMI : trop lent
    - possibilité pour l'avenir : EmbeddedJava --> version légère de RMI

Pourquoi HTTP entre agent et gestionn. ? (2/2)

- HTTP > socquettes :
  - communication naturelle entre servelettes au sein du serveur de gestion
  - même technologie utilisée au sein du serveur de gestion et entre agents et serveurs
  - config. du coupe-feu plus aisée pour des PME
    (par ex. serveur Web = serveur de gestion)

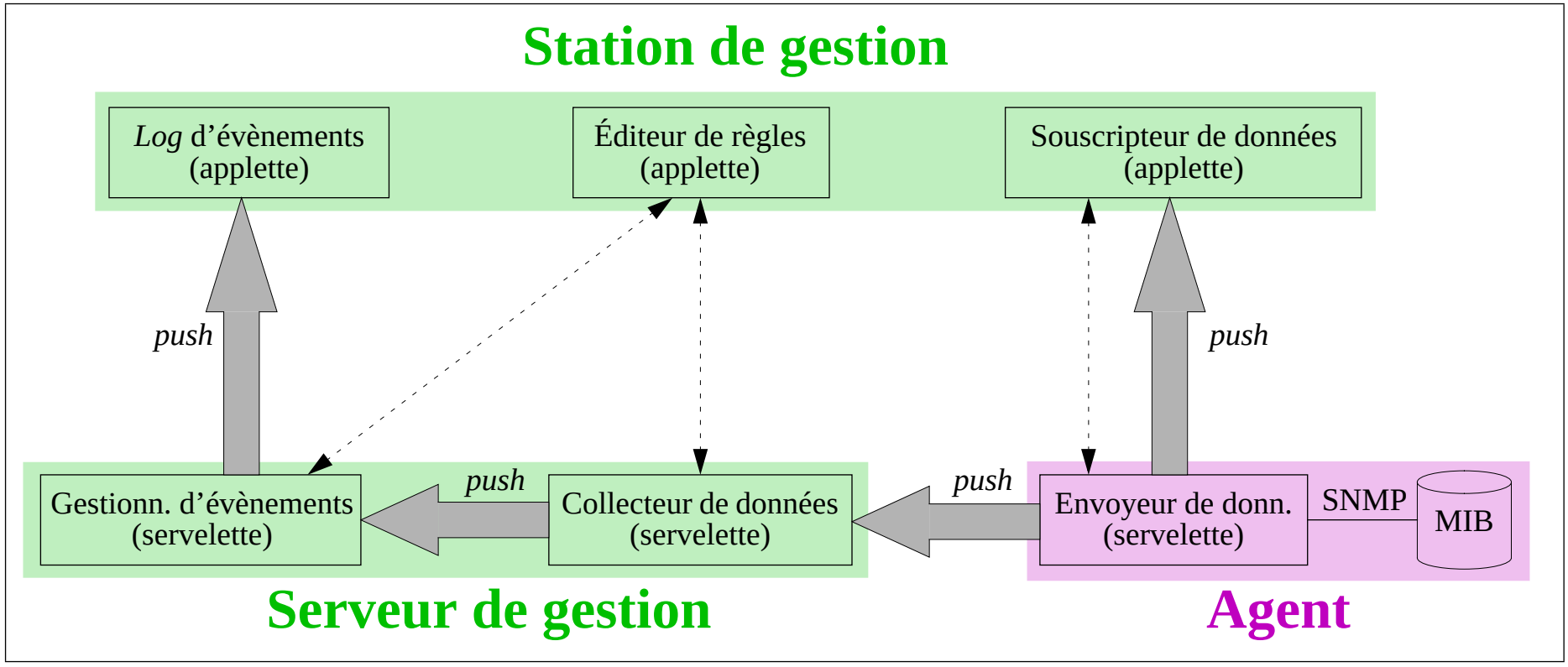
Pourquoi du *push* ?

- Économie de BP : réduction de l'*overhead* réseau
- Exemple : taux d'erreurs du trafic entrant pour l'interface n^{o} 3



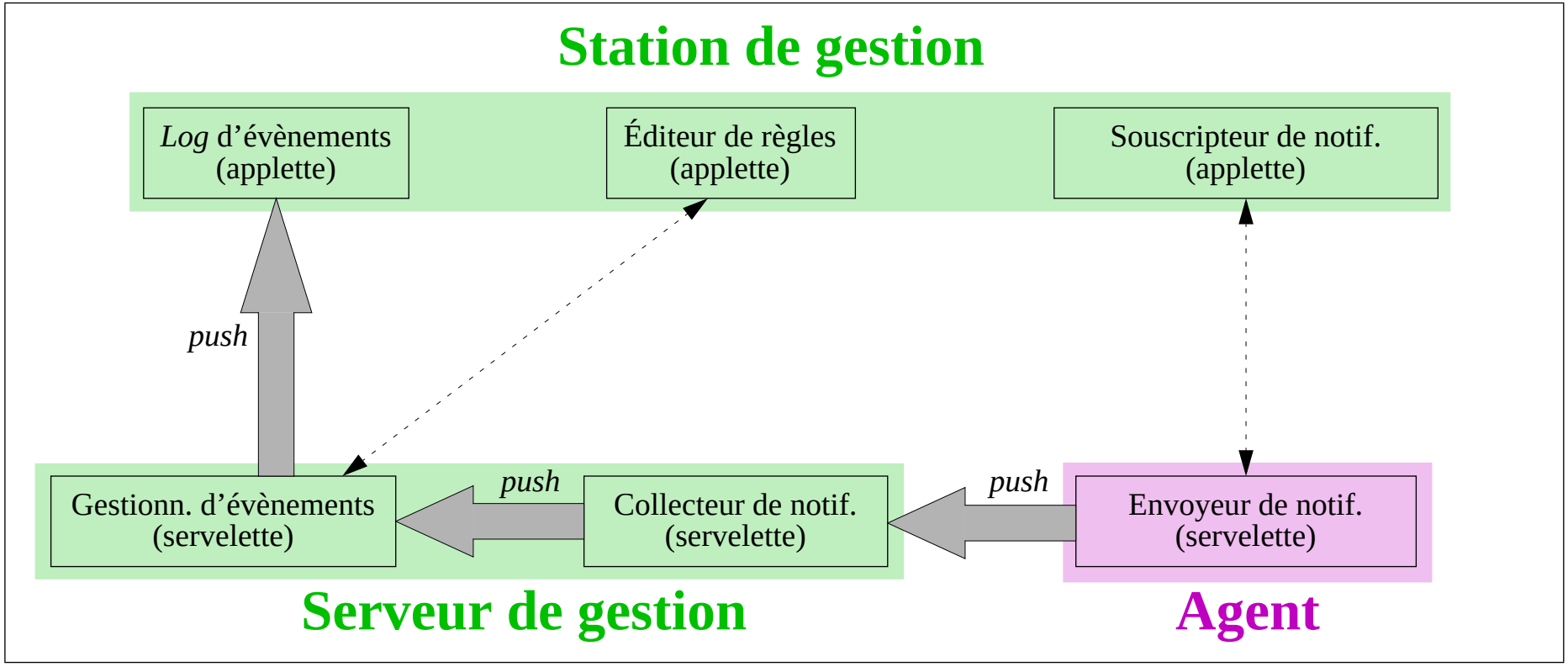
- Transfert d'une partie du travail du gestionnaire vers les agents
- 1ère étape vers la gestion par délégation (MbD) :
  - 2ème = délégation du prétraitement des variables de MIB aux agents

JAMAP : *monitoring* et collecte de données



JAMAP = JAva MAnagement Platform

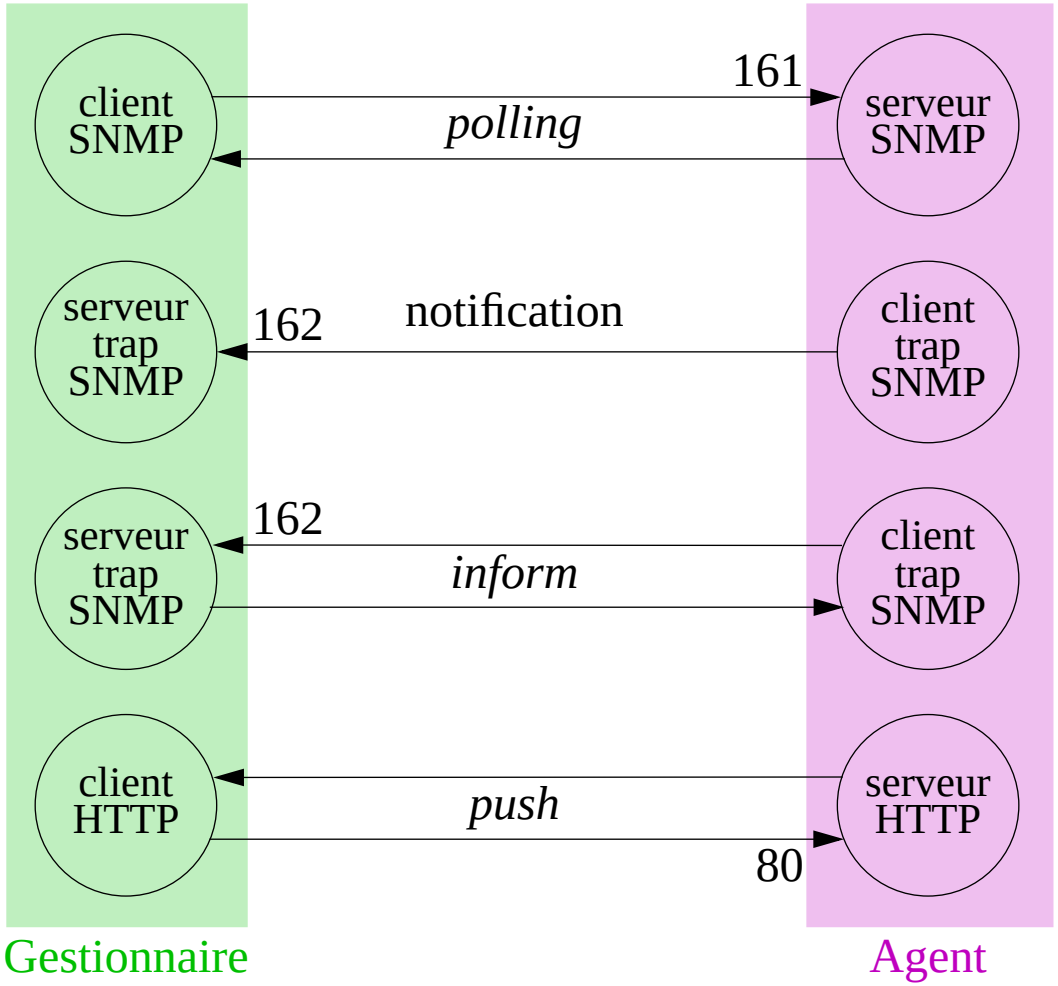
JAMAP : notifications



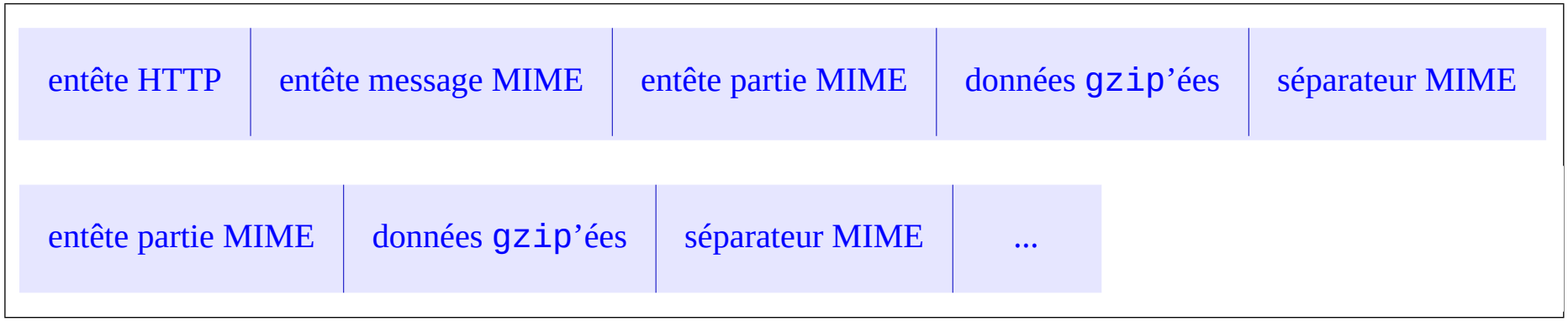
Problèmes

- Coupe-feu : la connexion doit être créée par le gestionnaire interne, pas par un agent externe
- Pour assurer la persistance des connexions :
  - les agents doivent pouvoir contrôler la tempo. de leur serveur HTTP
  - le gestionnaire doit se reconnecter à un agent quand la connexion est rompue
- Les positions du client et du serveur sont maintenant inversées :
  - le transfert des données de gestion est maintenant initié par l'agent
  - le côté client de la connexion persistante demeure côté gestionnaire
  - dans une architecture client-serveur, on veut que ce soit le serveur qui initie un transfert de données !

Positions du client et du serveur inversées



HTTP et MIME



MIME = *Multipurpose Internet Mail Extensions*

- Avantages :
  - simple à implémenter
  - coupe-feu : changement mineur (en supposant accès Web préalable)
- Inconvénients :
  - le gestionn. doit détecter toute coupure réseau et se reconn. à l'agent :
    - ➠ envoi de *keepalives* après 9 minutes sans recevoir de données
    - ➠ aveugle durant 9 minutes, ou envoi plus fréquent de *keepalives*

Conclusion (1/2)

- Que gagne-t-on à passer du *pull* SNMP au *push* Java pour gérer des réseaux IP ?
  - Plus besoin d'un PGR coûtant très cher
  - Utilisation de technologies Web standard plutôt que de SNMP qui est spécifique à un domaine donné --> compétence plus facile à trouver
  - Réduction de l'*overhead* réseau dû au transfert des données de gestion
  - Réduction importante du coût de développement des *add-ons*
  - Réduc. à zéro du temps de mise sur marché des *add-ons* --> *embedded*
  - Meilleure compétition entre petits et grands fournisseurs pour l'intégration de la gestion de réseau (plus besoin d'accords bilatéraux)
  - Simplification de la gestion de bureaux distants à travers un coupe-feu
  - Meilleur support de BD rel. tierce-parties
  - Compatibilité avec les équipements existants via des *proxies*

Conclusion (2/2)

- Que coûte le passage du *pull* SNMP au *push* Java pour gérer des réseaux IP ?
  - les fournisseurs doivent ajouter du logiciel dans leurs équipements :
    - ➠ un serveur HTTP (souvent déjà le cas aujourd'hui – cf. Networld+Interop)
    - ➠ un système de *push*
    - ➠ un ordonnanceur
    - ➠ un JDK et une JVM récents
  - les administrateurs doivent synchroniser les horloges des agents et des gestionnaires (par ex. avec NTP = *Network Time Protocol*)
  - le marché doit offrir des logiciels de qualité professionnelle pour le gestionnaire (servelettes Java) :
    - ➠ signe prometteur: de plus en plus d'acteurs dans le marché du *Web-based management* (cf. IM'99)