SeAl : Managing Accesses and Data in Peer-to-Peer Sharing Networks

Nikos Ntarmos Peter Triantafillou

Peer-to-Peer Information Systems

Tutor: Peter Triantafillou
Speaker: Konstantin Halachev
halachev@mpi-sb.mpg.de

Overview

1. Introduction
   a. Problem Overview
   b. SeAl
2. High-Level View of SeAl
3. The SeAl Monitoring\Accounting Layer
4. SeAl Verification Layer
5. Experiments and Performance Results
6. Conclusions

1.a Problem Overview

- The potentially huge number of users results in a great variance in the behavior
- No central authority to manage storage and computational resources
- P2P systems rely on the idea that peers are willing to share content/resources with the society. Peers are assumed to be altruistic
- Peers tend to have the most selfish behavior the system accepts. Peers use all the freedom they have, to be selfish
- The selfish behavior problem is crucial for performance, scalability and stability.

1.b SeAl

- Infrastructure transparently weavable into P2P sharing networks (structured and unstructured).

- Provide system with possibility to categorize peers and allow a regulated access to the resources depending on their contribution to the society

- SeAl manages the service peers receive depending on their contribution to the society and thus urges peers to be altruistic

Overview

2. SeAl Structure

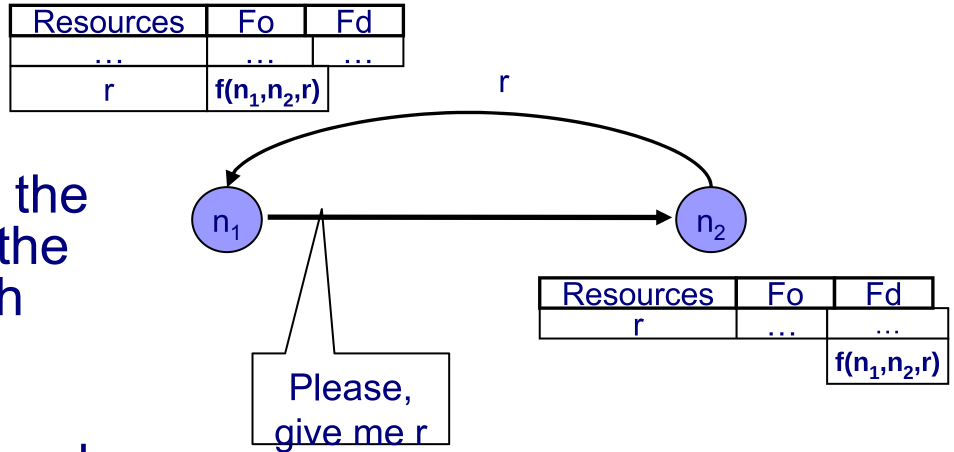
- Seal consist of two distinct layers:
  - SAL - SeAl monitoring\accounting layer
  - SVL - SeAl auditing\verification layer

- For simplicity reasons we assume operation in the context of file-sharing application

- SeAl counter-selfishness mechanism is based on the notion of favors

2.a Favors

- Peer n_1 owes peer n_2 a favor $f(n_1,n_2,r)$,when n_1 accesses a resource r shared by n_2
- Each peer keeps data about his favors in two lists
  - F_o – favors owed
  - F_d – favors done
- Ideally each peer will contribute the amount of content they take,so the system will be in equilibrium with $n_i.F_d=n_i.F_o$
- With this in mind we define selfishness as a function of F_ds and F_os,using $|F_d \backslash F_o|$ or $|F_d| - |F_o|$.



2.b Basic Notation and Infrastructure

- Independently of the underlying network SAL deploys a Distributed hash table (DHT) of its own, for its specific operations
- Every node in SeAl has a public\private key pair {n.kp,n.ks}.We assume that public keys of all nodes are accessible for every node
- We assume that every resource ,transaction are identified by a unique ID.A node ID is the hash of its public key
- Nodes are thus prevented from choosing their ID, because they have to prove the correctness of the public key they share

Overview

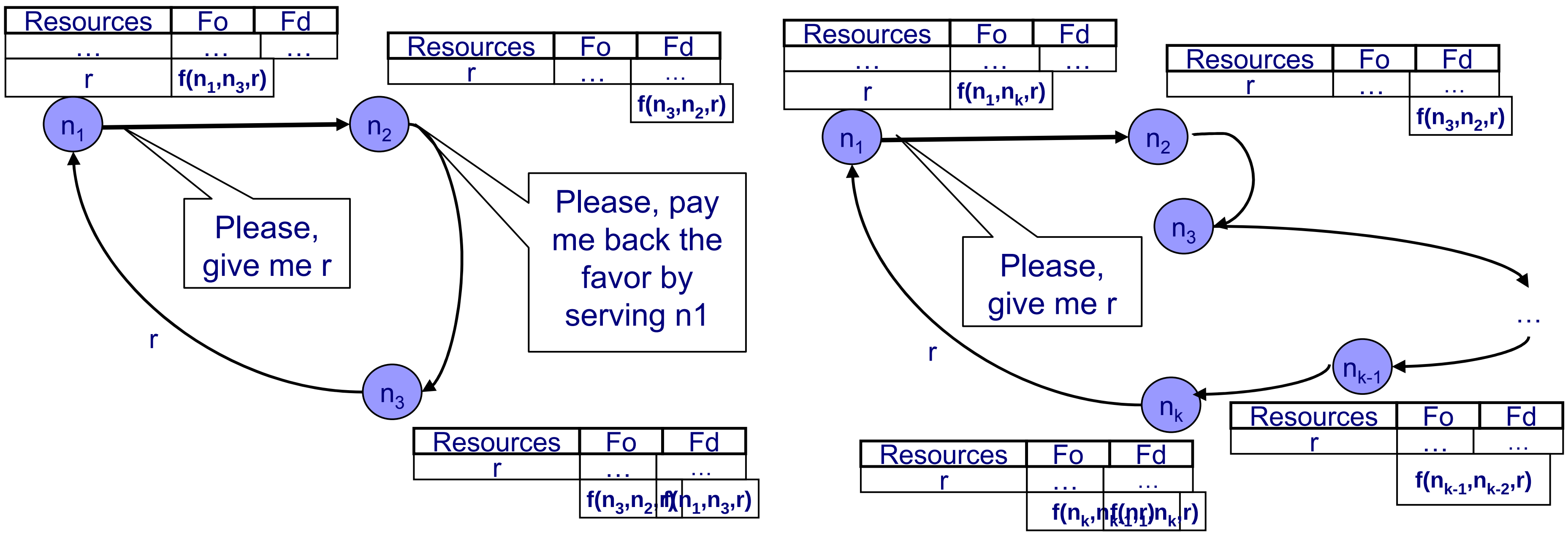
1. Introduction
2. High-Level View of SeAl
3. The SeAl Monitoring\Accounting Layer
    a. Transaction receipts and favors
    b. Favor Payback
    c. Bad reputation – the “black lists”
    d. Request scoring – the “white lists”
    e. Request serving –the incentives
    f. Debt Payback
4. SeAl Verification Layer
5. Experiments and Performance Results
6. Conclusions

3.a Transaction receipts and favors

- Each transaction in SeAl terminates with both sides having a digital “receipt” called “Transaction Receipt” – TR
- We denote by $TR(n_1.id,n_2.id,r.id,t)$ a Transaction Receipt concerning resource *r* being send from n_2 to n_1 at time *t*
- Favors in SeAl are implemented using TRs. Thus an entry in F_o or F_d is in the form $\{n_2.id,r.id,t,TR(\ldots)\}$
- A favor has a value of TR.r.size x TR.t\current time

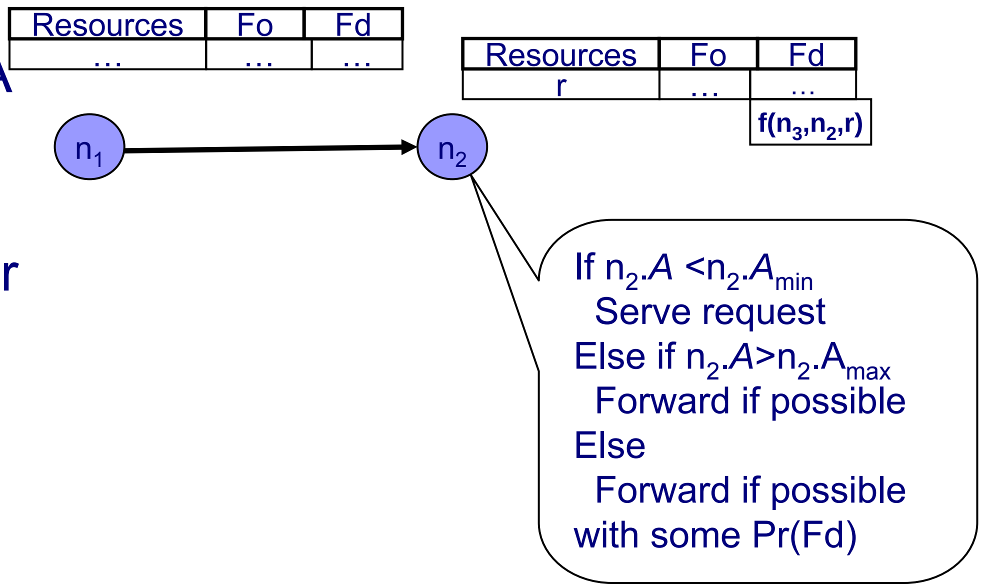
3.b Favor Payback

- Forwarding request to peers who owe a favor.



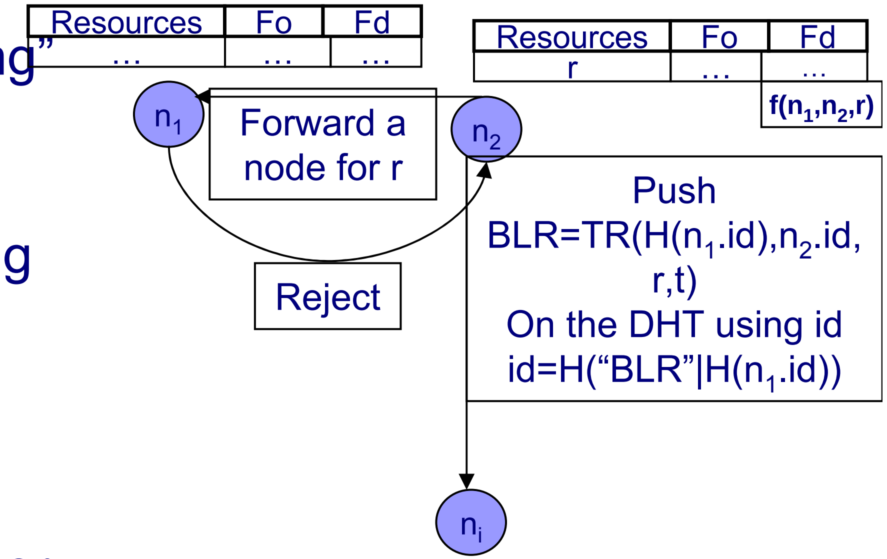
3.b Favor Payback

- Peers keep in track their altruism/selfishness score $n_i.A$
- Node administrators choose the formula for A as $|F_d|-|F_o|$ or $|F_d|\backslash|F_o|$
- They also choose a threshold bounds for A: A_{min} and A_{max}
- The decision on redirecting is based on these values



3.c Bad Reputation – the “black lists”

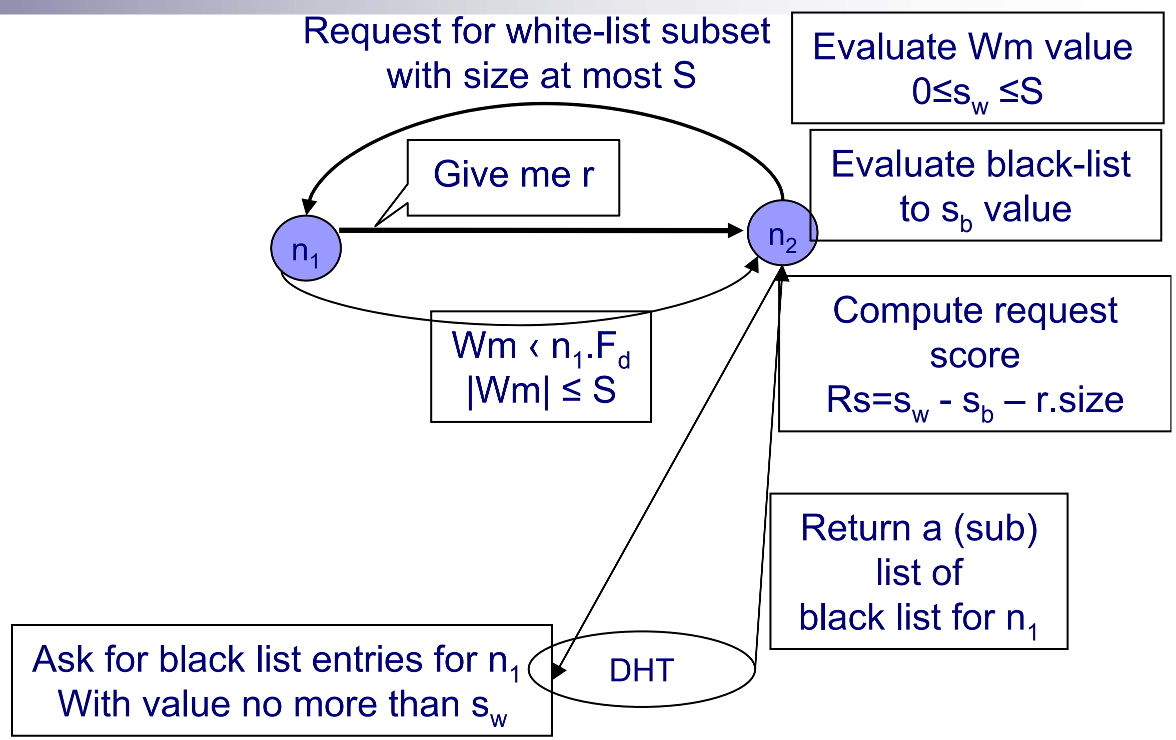
- Any deviation from the normal behavior may trigger the “black-listing” of the corresponding peer.
- A BLR is published on the DHT using an ID of H(“BLR”|H(n_1.id)),thus the node who stores the ID does not know for whom does it blacklist
- We treat the BLR objects as any other objects in DTH, thus storing and retrieving a (set of) BLR needs as many hops as the underlying DHT



3.d Request Scoring – the “white lists”

- When n_1 contacts n_2 about resource r
- n_2 asks for a proof that n_1 has done some good deeds
- Then he checks if he was blacklisted
- Final score of the request

  s_w-s_b-r.size



3.e Request Serving – the incentives

- For every incoming request :
  - First the request score is computed
  - Then it is stored in a sorted manner in the waiting queue
- Based on local decision a low-value request can be either scheduled for processing, allocated limited resources or even rejected
- Thus introducing an incentive of user to be altruistic

3.f Debt Payback

- Peers can regularly check the system for any black-listings against them
- If such exist they can contact the node that blacklisted them and offer to pay-back the favor.
- If all goes well the black listed node receives a TR denoting that it has paid its debt
- Then using this TR it can request the node storing the BLR to remove it from the network.

Overview

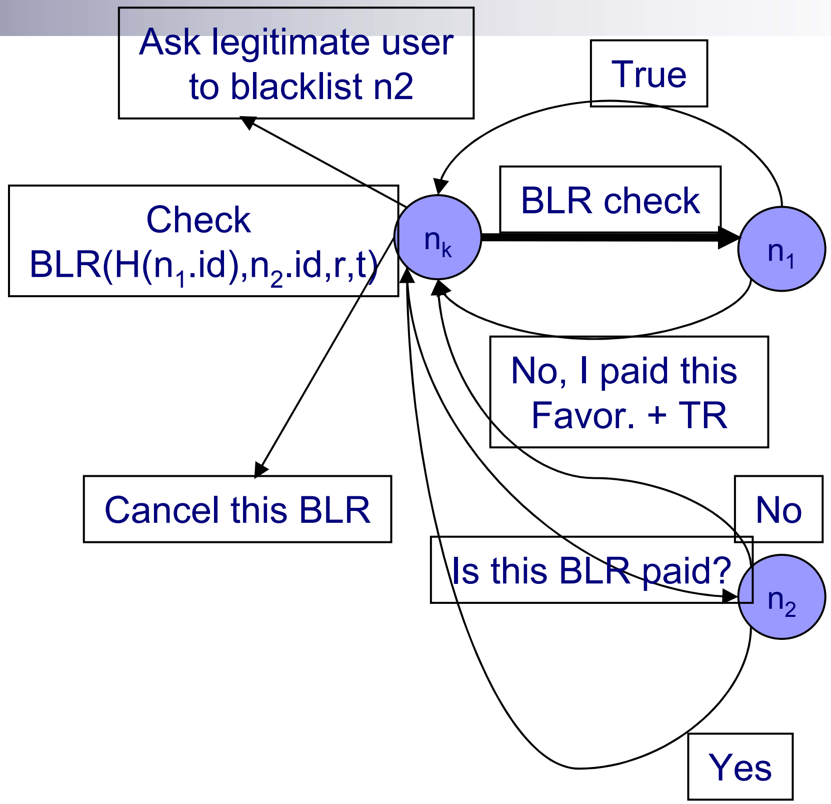
1. Introduction
2. High-Level View of SeAl
3. The SeAl Monitoring\Accounting Layer
4. SeAl Verification Layer
    a. Transaction receipt revisited
    b. Blacklists revisited
    c. White lists revisited
    d. File transfer
    e. SVL achievements
5. Experiments and Performance Results
6. Conclusions

4.a Transaction Receipts revisited

- How is TRs protected?
  - TR is signed by both participating peers
  - Each TR has 2 copies
  - Each third party can verify a TR by checking both signatures signed the receipt
  - If the verifier wishes it may even ask the serving node for the hash of r and thus check the resource specific info in r

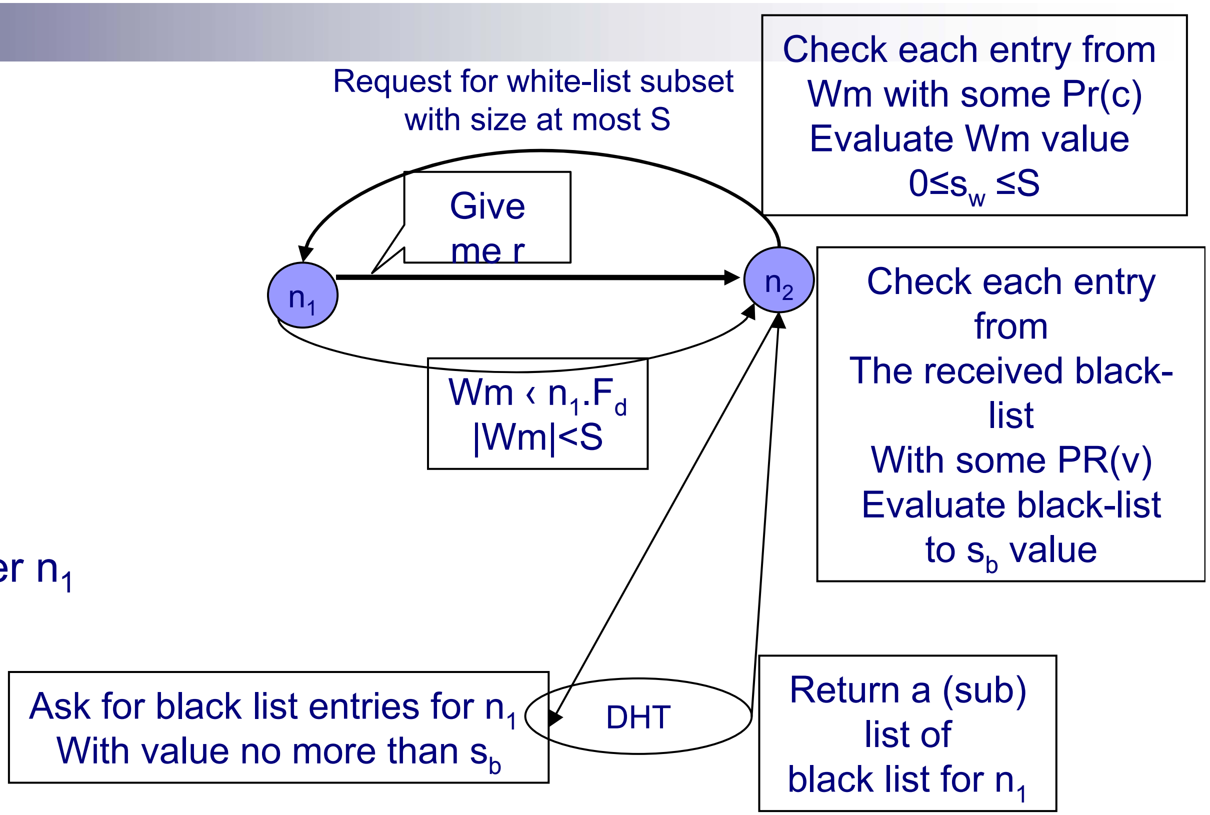
4.b Blacklists revisited

- A BLR cannot be forged
- A node receiving or retrieving a BLR can check it with Pr(v)
- Verification is done by asking the black-listed peer (and optionally the black-listing peer)
- Legitimate users may initiate the black-listings of the “perjurers”



4.c White lists Revisited

- Each white-list entry is verified with Pr(c)
  - Ask the served peer
  - Eventually check the resource
- Each black list entry is verified with Pr(v)
  - Ask the black-listed peer n_1
- This verification scheme introduces a trade-off between extra network accesses and possibly fake list



4.d File Transfer

- Server creates 2 symmetric keys
- Server encrypts r and one of the keys and send data to n_1
- n_1 creates initial draft of TR, signs it and sends it
- Then n_2 signs and sends the real TR and the decrypting key

Algorithm 1 File Transfer. Algorithm runs on $m.n_{server}$, unlessstated otherwise.

Require:
$send(msg, node\ ID)$: Send msg to node with given ID.
$\mathcal{E}_k(\alpha)$: Encrypt α using key k.
$\mathcal{S}_k(\alpha)$: Sign α using key k.

process(Msg m **)**

1: Generate k_1, k_2 = random symmetric-cipher keys;
2: $r_e = \mathcal{E}_{k_1}(r)$; $k_1' = \mathcal{E}_{k_2}(k_1)$;
3: $send(\{r_e, k_1'\}, m.n_{client}.id)$;
4: $m.n_{client}$:
  4.1: construct $TR' = \{m.n_{server}.id, m.n_{client}.id, r.id, t\}$;
  4.2: $TR_s' = \mathcal{S}_{m.n_{client}.k_s}(TR')$;
  4.3: $send(TR_s', m.n_{server}.id)$;
5: Verify the signature in TR_s';
6: $TR_s = \mathcal{S}_{m.n_{server}.k_s}(TR_s')$;
7: $send(\{TR_s, \mathcal{E}_{m.n_{client}.k_p}(k_2), m.n_{client}.id\})$;
8: $m.n_{client}$: recover k_2 and k_1 and decrypt r;
9: $F_d.add(TR_s)$; $m.n_{client}$: $F_d.add(TR_s)$;

4.e SVL Achievements

- The above scheme provides a strong disincentives but still not a complete solution to the common problems of Sybil attack and colluding peers
- Sybil attack is made undesirable because a peers looses its white list and thus it can gain a status a good as before
- Collusion attack is made undesirable because of the threshold of the white list you show
- Still both of the attacks have effect when they are combined
- With which we state that even with SeAl collusion in P2P networks is still an open problem

Overview

1. Introduction
2. High-Level View of SeAl
3. The SeAl Monitoring\Accounting Layer
4. SeAl Verification Layer
5. Experiments and Performance Results
   a. Test Models Setup
   b. Results and discussion
6. Conclusions

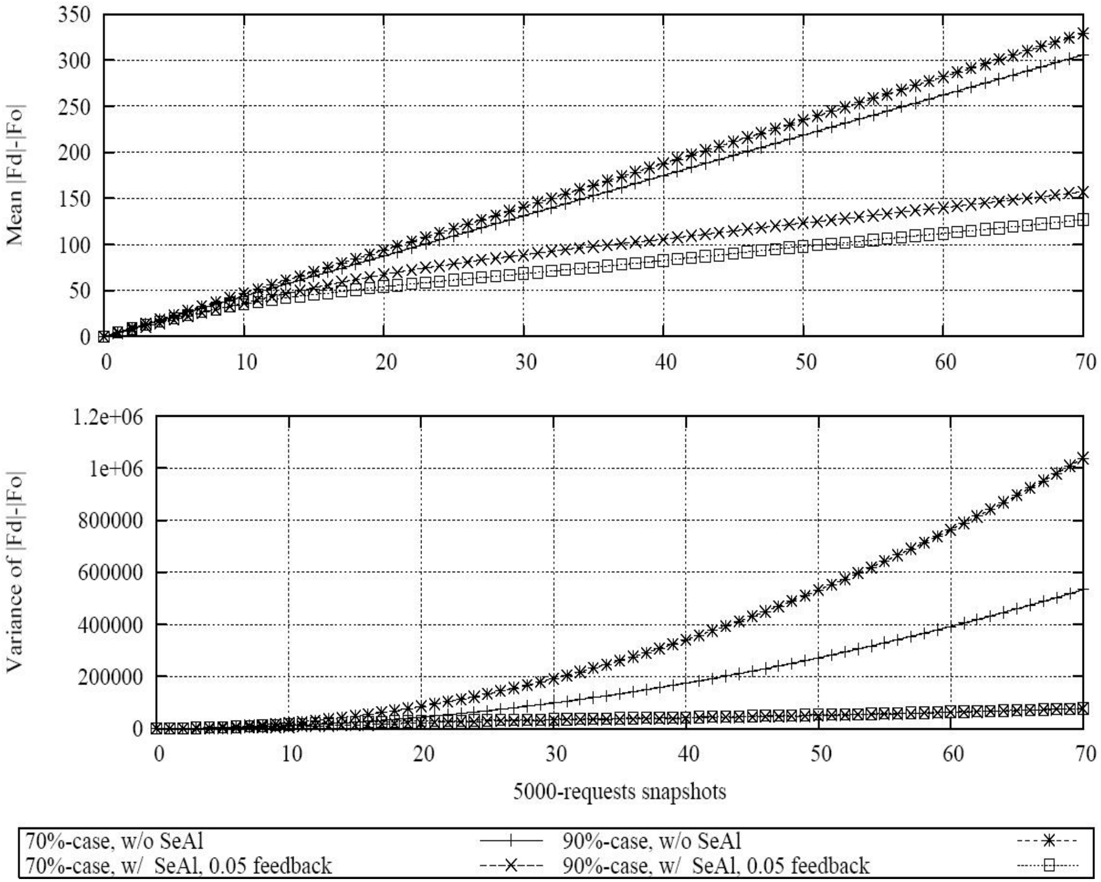
5.a Test Models Setup

- We assume a music-file sharing network with 50000 distinct documents of sizes 3-10 MB (average size of 6.5MB)
- 2048 peers
- Simulation of 1,000,000 requests following Poisson distribution, such that every peer will make approximately 5 requests a day of simulated time
- The peer population consists of 90%(70%) free-riders and 10%(30%) altruists with network connections from 33.6kbps(modem)-256kbps(cable) for selfish and 256kbps(cable) - 2Mbps(T1) for altruists
- Peers compute scores by $|F_d|-|F_o|$, Peers forwards requests with Pr 0,0.5 and 1
- Furthermore we have user behavior modeling values
  - Remain Altruist Pr(Ra)=0.8
  - Remain Selfish Pr(Rs)=1
  - Erase file Pr(Ef)=0.2, transfer abort Pr(Ca)=0.1
- If a request is delayed over some threshold we assume that user considers improvement in its behavior with probability Pr(Sd) improves its altruism probabilities by SD. We tested with Pr(Sd)=0.5 and SD=0.05

5.b Results and discussion

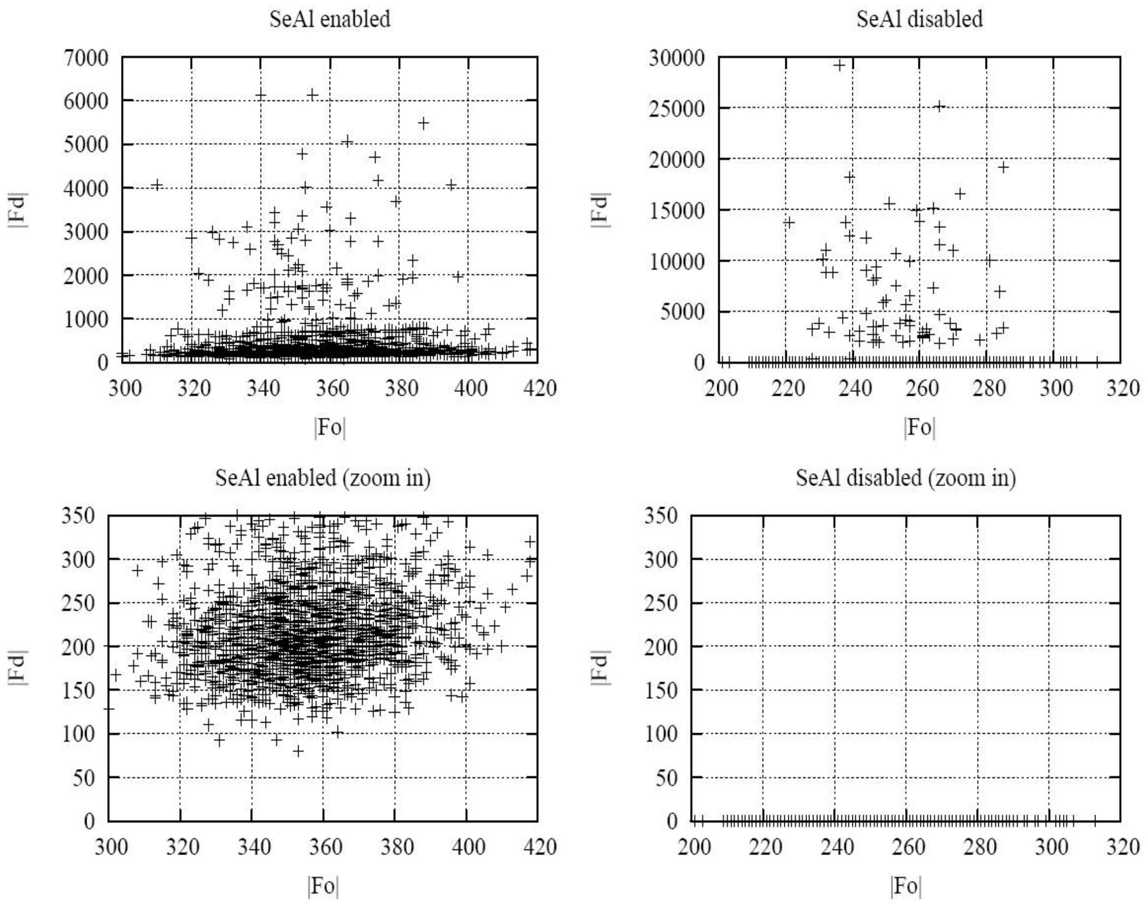
- We can observe that SeAl enabled applications have better mean and do not allow a great variance in the Altruism

- The network overhead caused by SeAl is only 0.4% of the overall network traffic



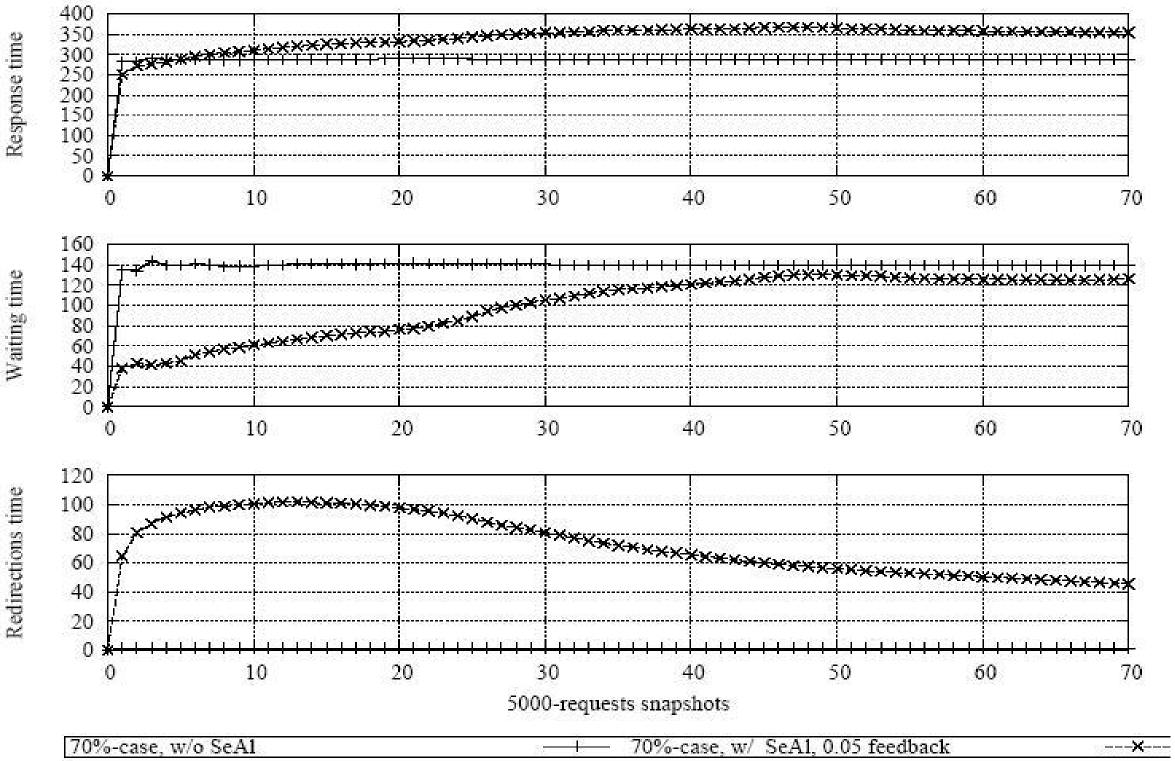
5.b Results and discussion

- We see 2 main clusters, for both of which we mark improvement
- Altruists are not so loaded
- And the selfish users have lifted the number of favors they do by a significant number of 200Average



5.b Results and discussion

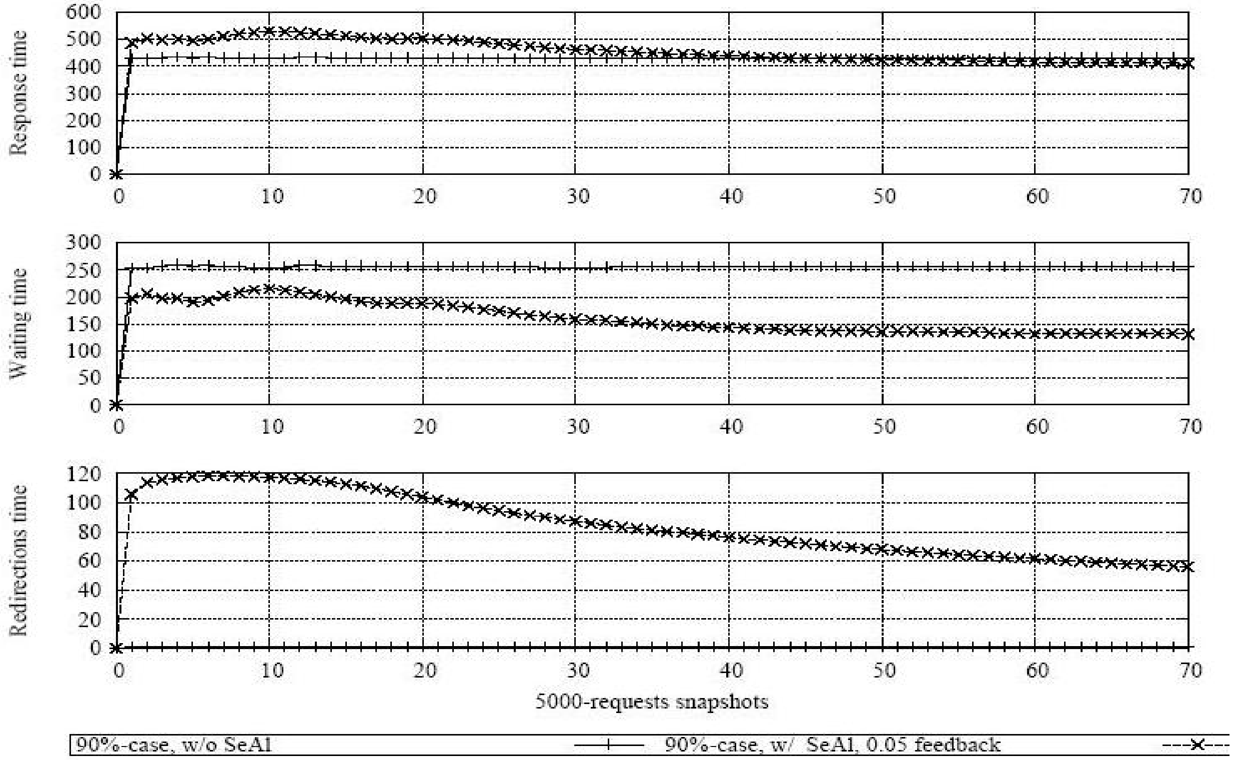
- Because of the large number of altruistic user we result in a 20% increase in the response time in the SeAl case



(a) 70% selfi sh user population

5.b Results and discussion

- However in this case, even with counting all the redirections the overall response time resulted in the SeAl system is lower and we have a better load-balance



(b) 90% selfi sh user population

Overview

Conclusions

- SeAl defines metrics of selfishness/altruism
- SeAl provides 2 subsystems which enable efficient ,auditable identification of selfish peers
- Still each peer can define its own selfishness limits
- Network, storage and response time overheads are measured to be very small
- Still SeAl does not offer a complete security solution but limits the influence of the Sybil attack and colluding users on the network.
- SeAl forms a complete infrastructure software layer for both structured and unstructured P2P network which makes it usable as a basis for development of wide variety of services in P2P networks

Thank you for your attention!