Personalized Negotiation Based on Individualization: Incorporating Personalization into Peer-to-Peer System

Javed I Khan  
*Kent State University*  
javed@kent.edu

Kailas B Bobade  
*Kent State University*  
kbobade@kent.edu

Manas S Hardas  
*Kent State University*  
mhardas@kent.edu

Abstract

In peer-to-Peer system user has control over what information about her could be released; however, inherent untrustworthiness of peers precludes her from deciding when, how and to what extent such information could be released. We have investigated a framework where users can personalize their information release policies. As opposed to simple request-response based communication, such individualization inherently necessitates a mechanism of negotiation. This mechanism of negotiation helps user to establish partial or full trust and based on these trust levels, user can release necessary information. To facilitate such individualization, we have presented negotiation enabled framework. In this paper, we provide extension to this work which facilitates selection of negotiation flavor on per-need-basis i.e. Personalized Negotiation. This is supported by negotiation protocol which defines the ordering of the messages and unique message structure that carries negotiation information.

Key Words- Peer-to-Peer, Personalization, Negotiation, Privacy, Security

1. Introduction

Peer-to-peer [6] system is referred as a network where all the nodes have same role and the system as a whole can operate without any specific centralized administrative capabilities. Every node can assume the role of a server which serves resources as well as client which access those resources without any authentication. Also, system allows nodes to join or leave network freely. P2P is currently used by millions of people across the globe to share information like voice, video, text etc. Unfortunately, the access control for P2P systems to-date is rudimentary.

In this paper we demonstrate a new approach for access control of the resources suitable for P2P systems. In this approach conforming to P2P principle, instead of any centralized entity, P2P clients empower individual users to specify their personal information and credential release policies. Once release policies are established, P2P client work as negotiation agent on behalf of the users and the network infrastructure acts as a mere conduit of communication. It keeps end-user at the top of decision process- deciding exactly how their information, credentials, and resources might be released and handled. Such personal control of personal information is so important that indeed some researchers believe that individual’s control is fundamental to the notion of privacy- or in other words, without personal control over the personal information release policy, a system cannot claim to provide privacy (Alan Westin [5]). It provides an interesting and new approach in looking into the access management issues in large networked societies.

In real-society any release is generally associated with reciprocal release from the other side. So any release action in real life is generally a negotiation process. The release policy and corresponding negotiation process in real life requires individualization. This is due to the intrinsic variability of circumstances under which information is exchanged.

Let’s consider some real-life scenarios explaining personalization. Consider the case of curriculum-vita (CV). To apply for job, every manager request information like Transcripts, SSN, and email from students. How a student would release CV information to companies can be bewildering. For example, a student may want to share his detail grade from transcript only to companies with a job offer in specific area within specific region. But, he may not want to release it indiscriminately to any head-hunter. Yet another student may want to release her cell phone and personal contact details fields only to a company HR which has specific interview offer. Yet another student who is already employed may want to restrict his specific identity and employment history fields online unless he receives the specific name of the company or are sure that the potential employer is safely a company from another state. Thus release policies are seldom universal. On the other side, every employer
might have personalized release polices for job information. These personalized policies, at both ends, trigger credential exchange which results in negotiation.

Although personalized negotiation is the generalized form of all mutual transactions - be it financial, P2P, barter exchange, or identity management, we will focus on personalized negotiation in P2P systems. Consider a private Peer-to-Peer network formed by group of Universities and Companies. A prototype of this setup is shown in fig 1. In this setup, university has a peer called Locally Trusted Third Party (LTTP) - blue circle - which issues credential for every piece of registered student’s information. Also company’s LTTP - black circle, issues credential for every piece of her manager’s information. Once credentials are issued, LTTP doesn't interfere with transaction between peers.

In [4], we presented a negotiation enabled framework. This framework is also applicable in peer-to-peer system considering that framework is available for every client. In this paper, we extend that framework to facilitate selection of negotiation flavor on per-need-basis i.e., personalized negotiation. For example users might be interested in less number of communication steps or disclosing only necessary credentials or releasing credentials only when it becomes clear that successful negotiation is possible. To accommodate most of these likely requirements, we are presenting a negotiation protocol in section 2 and algorithm in section 3.

Previously, [1] proposed model to authorize action on private information. But it didn’t provide interaction between policy holders – an inherent need of Individualization. [3] has proposed an approach for negotiation, using third party, without considering Individualization.

2. Negotiation Protocol

This section provides a prototype of protocol for bipartite negotiation. Protocol determines the sequence of messages and a message structure carrying negotiation information.

2.1. State Transition Diagram

A negotiation proceeds through six states namely Advertisement, greetings, strategy, active, negotiation, and adieu shown in figure 1. Important messages used in negotiation are Greeting, Advertisement, Solicitation, Strategy, Negotiation, Deal, No_Deal and Reporting.

Negotiation moves from start to greeting state, when one party invites other for a negotiation session by offering initial identity. After sending greetings message, each party waits for greetings message from the opposite party. Opposite party can accept or reject the invitation (and negotiation proceed to adieu state). If both parties agree then negotiation enters into active state. Here, negotiation can also moves to advertisement or strategy state. In advertisement state, a party sends advertisement messages naming Target Resources and waits for the other party to reciprocate by sending solicitation (advertisement) message. If one party is interested to offer the Target Resource and other party is interested to obtain it then negotiation moves to active state otherwise moves to adieu state if any one of them is not interested.

In strategy state, a party accepts list of negotiation strategies from opposite party, and if both parties agree on a strategy then proceed to active state (or jump to adieu state). Negotiation moves from active to negotiation state when both parties fulfilled prerequisite like receiving each other’s identity or deciding strategy or deciding resource to negotiate. In negotiation state, actual negotiation starts by sending negotiation message and protocol moves through series of exchanges until negotiation becomes successful or unsuccessful. After this, negotiation moves to adieu state. In adieu state both sides perform reporting about current negotiation session and then moves to the end state. Negotiation protocol moves to end state if any error condition occurs at any given state.
2.2. Negotiation Message

Negotiation Message has two main compartments - header and body. Header contains Action, Session Id, Strategy, and Security fields. Strategy field carries negotiation strategy chosen by Negotiation Initiator and Negotiation Responder. This field is useful when two parties negotiate for negotiation strategy. Security field carries encrypted identities of participant organizations, and members. Action field contains one of the messages described in section 2.1.

Body has Initiator’s Resource List (IRL) and Responder’s Resource List (RRL). Each item in these lists has the following fields. Resource Descriptor (RID) is the name of resource to uniquely identify it. The Value field contains the value or an URL to the value of the resource. The type field of a listed resource can have one of the following values namely Personal (P), Credential (C), Attribute (A) or Information (I). The License field contains the granted disclosure policy for the resource. State field contains current state of negotiation of a resource during negotiation process. Extra field carries state information of every resource involved in negotiation, if necessary. Previous two fields achieve stateless resolution process i.e. Negotiation parties wouldn’t require maintaining complex state information.

3. Stateless Proxy Attribute Negotiation

We have presented one flavor of negotiation resolution algorithm i.e. Stateless Eager Attribute Negotiation (SEAN) in [4]. Here, we will discuss Stateless Proxy Attribute Negotiation (SPAN) which is based on PRUNES [2]. Above mentioned negotiation message can accommodate SEAN as well as SPAN.

In the first phase of proxy negotiation, two sides exchange the resources’ name to let each other know their requirement constraints without disclosing resources values. During this phase both parties track the requirement dependencies. Here, goal is to determine if the deal is possible and if it is, only then they perform the actual resource exchange in second phase. Based on tracked requirement dependencies in the first phase, both parties exchange resources using optimum sequence in second phase. We describe this algorithm next.

3.1. Driver Process

The input to SPAN_State_Machine, in figure 3, is the responder’s release policy and the newly received message from Initiator. It’s output (line 22) is four sets-(1) those ready to be released (NEW_RELSET), (2) pending (NEW_PENSET), (3) those he would like to deny (NEW_DENSET), and (4) a list of new resources he would like to counter request (NEW_REQSET).

```python
# SPAN_state_machine(next, RRL) {
1. OLD_RELSET - {AAP, A, state - REJECTED}
2. OLD_REQUEST - {AAP, A, state - AVAILABLE}
3. OLD_PENSET - {AAP, A, state - PENDING}
4. OLD_DENSET - {AAP, A, state - DENIED}
5. init - REQUEST = 0
6. meta - state = CONSTANT
7. (AAP, A, C, state - RELEASED) and (AAP, A, C) 
8. rea = REQUEST
9. case (0), (0), (0), (1) - meta- state
10. next
11. (AAP, A, (AAP, A, C) state - RELEASED) and (AAP, A, C)
12. rea = REQUEST
13. return (0), (0), (0), (0) - meta- state
14. Else

15. For each AAP, A in OLD_RELSET
16. (AAP, A, C, REQUEST = SEAN_request(AAP, A, C, REQUEST, AAP))
17. NEW_REQUEST = (NEW_REQUEST) U (AAP, A, C, REQUEST = REQUEST + 1) 
18. NEW_RELSET = (NEW_RELSET) U (AAP, A, C, REQUEST)
19. NEW_REQUEST = (NEW_REQUEST) U (AAP, A, C, REQUEST) 
20. NEW_PENSET = (NEW_PENSET) U (AAP, A, C, REQUEST) 
21. (AAP, A, C) state - DENIED
22. // End of for from line 15
23. NEW_RELSET = (NEW_RELSET) U (AAP, A, C, REQUEST) 
24. NEW_PENSET = (NEW_PENSET) U (AAP, A, C, REQUEST) 
25. // End of for from line 14
}
```
This routine calls the SPAN_Rule_Resolver routine (line 16) for each requested resource in the message list. Resolver routine determines if the resource could be released and if not, what other resources need to be counter requested, or if the resource has to be denied. Resolver routine’s inputs are (i) a resource in the set of old requests (OLD_REQSET), (ii) resource lists in the message, and (iii) the rules particularly linked to selected resource from old requests set. The routine changes the states of the resources and it generates new release set (RELSET) which contains resources that need to be released (by owner) and the counter request set (REQSET) which contains resources that need to be requested from opposite party.

Corresponding to the four new sets, SPAN_State_Machine maintains four copies of old sets (OLD_REQSET, OLD_RELSET, OLD_DENSET, and OLD_PENSET) extracted directly from an incoming message (lines 1-4). The routine checks (lines 11-13) old sets to see if the negotiation has resulted in a deal. Negotiation becomes successful if Target Resource is now available in OLD_RELSET. Negotiation becomes unsuccessful if Target Resource is in OLD_DENSET (lines 7-9). If above conclusions cannot be made, then the Resolver routine is called (line 16) to process every resource in OLD_REQSET. Resolver routine generates two sets - REQSET and RELSET. These new REQSET, RELSET provided by the Resolver routine and already existed old sets generate the new sets (NEW_REQSET, NEW_RELSET, NEW_PENSET, and NEW_DENSET) for the outgoing message (lines 17-21).

3.2. Resolver Process

SPAN_Rule_Resolver uses important variables such as 1) GARC (Global Attribute Release Count): Total number of resources released by both parties. 2) ARC (Attribute Release Count): When a resource is denied (by owner), its ARC is used to save the current GARC value. 3) CQ (Counter Request): Resource asked as a counter request for each pending resource. 4) LURA (Clause that causes release of attribute): A clause in a rule that has resulted in release of requested attribute and it is used to trace back the release sequence in second phase of proxy negotiation.

A rule in a policy is represented in the disjunctive normal form (DNF) as \( R_1 \leftarrow C_1 \lor C_2 \lor \ldots \lor C_r \), where, clause \( C_m = I_1 \land I_2 \land \ldots \land I_k \), which means resource \( R_1 \) will be released when either of clauses \( C_1, C_2, \) or \( C_r \) is satisfied. Here, clause \( C_1 \) requires all resources \( I_1, I_2, \) and \( I_k \) from the other side. Each rule and clause is processed from left to right.

Resolver routine considers requested resources one by one and the rules associated with that resource and processes as explained below: **STEP-1** If a requested resource doesn’t have CQ, means it is being requested for the first time, routine executes (lines 6-7, lines 15-21, lines 29-33) as explained next: If the first resource in requested resource’s first clause is not released and not pending, then this first resource will be counter requested (REQSET) and stored in requested resource’s CQ (lines 17-21). But if this first resource is already released (by opposite party), then the next resource in the same clause is counter requested and so on until all resources in one clause are exhausted. If all resources in one clause are released, which can be found out from Resource List available in the body of the message, then requested resource can be released. When released, GARC is incremented by 1 and current clause is saved in the LURA of the released resource (lines 29-33) **STEP-2** If a requested resource already has CQ, then routine executes (lines 6-12 and lines 6-21, 29-33) as explained next: If CQ of requested resource is still pending, then the routine will come out of the current rule and keep track of the same resource and the next resource from the same clause is considered for CQ and so on. **STEP-3** If after STEP-1 or 2, there is a new counter request and if this new counter request (which is different from old counter request, if any) is already pending, then it will skip that clause and move on to next

![Figure 4. Dependency Graph Generator and Offered Request routine in SPAN](image-url)
clause. This is because it indicates a cyclic dependency which cannot be resolved with current set of released resources. If rest of the clauses too has at least one resource whose request is already pending, then the requested resource is denied and the GARC is saved in its ARC (lines 34-35).

STEP-4) Before finally placing the counter requests, the routine will check if that resource has been denied by the other side. If this new counter request resource had been denied then (line 22-28, 34-35): 4.A) that resource become CQ only if it’s ARC > GARC (lines 22-26). This is because, there are more releases now since the last denial - so previous cycle may not be a problem anymore. 4. B) that resource doesn’t become CQ if it’s ARC = GARC. This is because nothing has changed since last denial. In this case, the clause containing that resource is skipped for next clause (line 28) to repeat STEP-1 or 2.

3.3. Release Process

At the end of successful proxy negotiation, message carries release clause for every negotiated attribute. Using this information each party generates logical dependency graph to find out sequence for exchanging resource’s value. Based on dependency graph, both parties release resources as explained by Offered_Set routine (in fig 4).

Offered_Set routine executing at each end of negotiation finds out which resources can be offered to opposite party using i) logical dependency graph and ii) resources offered by opposite party (i.e. old_offeredset) till that time of resource release phase, if any.

LDGraph_generator routine (in fig. 4) generates dependency graph using rlist as input. A node in dependency graph is a resource whose state of negotiation is offered or available and other nodes are solved clause of its disclosure policy (with one edge from each resource in that clause connecting to offered node). This way resources in dependency graph are represented (lines 2-9). At the start of second phase, dependency graph is generated from the resources whose state of negotiation is available because none of the parties have offered any resources yet. But once parties start offering resources to each other, they will move state of negotiation of the resources from available to offered, and then Logical Dependency Graph is generated from resources whose state of negotiation is either available or offered (line 1).

In offered_set routine, one of the parties will offer resources from dependency graph which don’t have disclosure policy (lines 3-7). After this, opposite party will offer his resource from dependency graph according to offered resources it has got from first party (line 8-14). Thus two parties keep on offering resources to each other till Negotiation Requester gets Target Resource.

4. Negotiation Example

We consider a job seeker-hunter scenario in a complex federation setup - group of Universities and Companies. Students store Individualized Policies with their universities and various managers in the companies set up their requirements to search potential employees. In this setup, we consider three students Alice, Pooja and Sajid with home institution KSU and three hiring managers Bob, John, and Yang from KLM Inc, ABC Inc, and DEF Inc respectively with release policies in figure 5. Here, we will discuss one negotiation in detail where Negotiation Agent of KLM Inc initiates a search for potential candidate (Pooja) to fill-up the vacancy under Bob. Before that, let’s understand state information carried by each negotiation message.

In our notation we group resources against their respective current states in a negotiation as STATEP ((R1:V1), (R2:V2).... (Rn: Vn)). STATE can be the states of a resources specified in [4] such as REQ, AVL, PEN, DEN, OFF etc. Each argument is a pair where Rn is the ID of the resource and Vn is the special information about the resource. The superscript P represents the party (negotiation initiator (I) or negotiation responder (R)) involved in the latest update of a state. Top portion and bottom portion of each message corresponds to IRL and RRL as described in section 2.2.

KLM’s agent starts negotiation by requesting Interview i.e. (REQ I (R1, Ø)) from Pooja’s agent in message 1 in fig 6 A).Here, Ø indicates that R1 has no counter request (CQ). Bottom portion of this message i.e. Pooja’s dataspace is empty. In response to message 1, Pooja’s agent will counter request Job_Title (REQ R (I3, Ø)) in message 2 and also changes state of negotiation of

<table>
<thead>
<tr>
<th>STATEP</th>
<th>STATEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R1:V1), (R2:V2).... (Rn: Vn)</td>
<td>(R1:V1), (R2:V2).... (Rn: Vn)</td>
</tr>
</tbody>
</table>

Pooja’s agent responds to KLM’s agent by counter requesting Job_Title (REQ I (I3, Ø)) in message 3. Here, Ø indicates that KLM has no counter request (CQ). KLM’s agent responds to Pooja’s agent by counter requesting Interview (REQ R (R1, Ø)) in message 4. Here, Ø indicates that Pooja has no counter request (CQ).
Interview from requesting to pending i.e. PENR \((R_1, I_3)\). Message 3 shows request of R7 i.e. REQI \((R_1, I_3), (R_7, R)\) and as per Pooja’s policy, she needs I3 to unlock R7. But state of negotiation of I3 is already pending and it requires R7 to unlock. This creates a cyclic dependency. To resolve this, Pooja’s agent will deny R7 i.e. DENR \((R_7, 0)\) in message 4. Here, 0 is GARC when R7 is denied.

In response to message 4, KLM’s agent will find out that R7 is counter request of I3. So even though R7 is denied, I3 can still be unlocked using R2. So KLM’s agent will request R2 in message 5. This process of searching an alternative path to unlock an attribute (I3 in this case) and thus breaking cyclic dependency is called as backtracking. To resolve this, Pooja’s agent will deny R7 i.e. DENR \((R_7, 0)\) in message 4. Here, 0 is GARC when R7 is denied.

In response to message 4, KLM’s agent will find out that R7 is counter request of I3. So even though R7 is denied, I3 can still be unlocked using R2. So KLM’s agent will request R2 in message 5. This process of searching an alternative path to unlock an attribute (I3 in this case) and thus breaking cyclic dependency is called as backtracking. In backtracking, a denied attribute is requested again only if number of attributes released by both participants (GARC) are more since last denial of the same attribute. As per this rule R7 is again requested by KLM’s agent in message 9. Here, GARC=1 in message 8 is more than GARC = 0 when R7 was denied in message 4.

Finally, in message 12 Pooja’s agent offers Interview i.e. AVLR \((R_2, R_3, I_3)\) to KLM’s agent to finish first phase of negotiation. In second phase, agents will exchange attribute values using dependency graph in figure 6 B). At the end of second phase, agents will generate negotiation result for their participants in 6 C).

5. Analysis of SEAN and SPAN

Figure 7 shows results of negotiation among three organizations, and three students (policies shown in fig 5) for SEAN and SPAN. In successful negotiation, SEAN releases more attributes and fires more number of rules than SPAN but with fewer messages. In unsuccessful negotiation (CDE-Pooja) SEAN releases few attributes, but SPAN doesn’t release any attribute. Here, users which are eager to reach a deal can opt for SEAN at the cost of disclosing more attributes while cautious users, with sensitive attributes, could opt for SPAN to avoid unnecessary disclosure of the attributes.

Communication Complexity of SPAN and SEAN [4] is shown in figure 8.

![Diagram of negotiation process](image-url)
6. Conclusion

In recent times P2P user privacy has been seriously compromised and so is the trust. Privacy and user information have been released in all forms - inadvertently, by way of theft or system mishaps. These threat moves users away from P2P system. So there is an urgent need to protect user privacy in this system.

As a step towards that we have contributed a novel negotiation enabled framework and protocol to enhance user privacy in P2P system. This framework facilitates users to choose negotiation flavor on per-need-basis i.e. personalized negotiation. Two such flavors are Stateless Eager Attribute Negotiation (SEAN) and Stateless Proxy Attribute Negotiation (SPAN). SEAN results in successful negotiation with minimum communication steps while SPAN discloses only necessary attributes for negotiation.

7. References