Firewall Policies Provisioning Through SDN in the Cloud

Nora Cuppens, Salaheddine Zerkane, Yanhuang Li, David Espes, Philippe Le Parc, Frédéric Cuppens

Firewall Policies Provisioning through SDN in the Cloud

Nora Cuppens[1], Salaheddine Zerkane[123], Yanhuang Li[1], David Espes[23], Philippe Le Parc[23], and Frédéric Cuppens[12]

[1] IMT Atlantique, 2 Rue de la Chataigneraie, 35510 Cesson-Sevigne, France
[2] BCOM, 1219 Avenue des Champs Blancs, 35510 Cesson-Sevigne, France
[3] Université de Bretagne Occidentale, Lab-STICC, 20 Avenue Le Gorgeu, 29200 Brest, France

Abstract. The evolution of the digital world drives cloud computing to be a key infrastructure for data and services. This breakthrough is transforming Software Defined Networking into the cloud infrastructure backbone because of its advantages such as programmability, abstraction and flexibility. As a result, many cloud providers select SDN as a cloud network service and offer it to their customers. However, due to the rising number of network cloud providers and their security offers, network cloud customers strive to find the best provider candidate who satisfies their security requirements. In this context, we propose a negotiation and an enforcement framework for SDN firewall policies provisioning. Our solution enables customers and SDN providers to express their firewall policies and to negotiate them via an orchestrator. Then, it reinforces these security requirements using the holistic view of the SDN controllers and it deploys the generated firewall rules into the network elements. We evaluate the performance of the solution and demonstrate its advantages.

Keywords: Security Policies, Software Defined Networking, Cloud Computing, Orchestration, Firewall, OpenFlow, Service Providers, ABAC

We implement our solution into an existing SDN Firewall solution by enhancing its orchestration layer. Then, we deploy a use case for our proposition in an SDN infrastructure. The scenario compounds a NSC and 3 different NSPs. Each provider delivers a type of SDN firewall whether it is a stateful proactive SDN firewall [36], a stateful reactive SDN firewall [37] or a stateless SDN firewall [16], [31]. All the peers express their firewall policies using our language. The Orchestrator mediates between them, selects the best provider and runs a negotiation process in order to reach a mutual agreement with the customer. Afterwards, it sends the contract to the chosen firewall service. The latter interprets it into OpenFlow [1], [12], [25] rules and installs them in the network elements. We evaluate the performance of our framework by setting a test-bed for the aforementioned scenario.

The rest of the paper is organized as follows: Section 1 reviews existing proposals on policy-related solutions. Section 2 describes the formalism and all the

processes from policy expression till the interpretation process of the firewall policies into OpenFlow rules and their deployment. Section 3 presents the integration of our solution into an existing SDN firewall environment and its performance experimentation results. Section 4 concludes the paper and outlines future work.

1 Related Work

The literature lacks of propositions that integrate to SDN security applications (especially SDN firewalls) firewall policy provisioning in the orchestration layer. There is an open research field on the subject in terms of SDN as a service and firewall policies orchestration in the cloud. To the extent of our studies the majority of SDN security solutions do not support negotiation between firewall policies, neither propose mutual agreement processes between providers nor reinforcement function of the client-provider agreement. CloudWatcher [32], FRESCO[33],[13] and OpenSec [17] are three famous SDN propositions that rely on specific policy script languages. They lack interoperability and openness since they are plateform specific. In addition, they do not integrate a policy management process in order to interact with the cloud level. Other solutions focus on policy expression and enforcement. Tang et al. [34] develop a service oriented high level policy language to specify network service provisioning between end nodes. Batista et al. [2] propose the PonderFlow, an extension of Ponder [9] language to OpenFlow network policy specification. EnforSDN [3] proposes a management process that exploits SDN principles in order to separate the policy resolution layer from the policy enforcement layer in network service appliances. The concept improve the enforcement management, network utilization and communication latency, without compromising network policies. However, it can not handle stateful security applications like stateful firewalls.

Many solutions have been proposed for selecting NSP in the cloud. There are two trends in the literature. The major one focusses on NSC's security requirements without taking into consideration NSP's constraints. [18],[23] define the selection strategy exclusively on NSC's capacity. Bernsmed et al. [4] present a security SLA (Service Level Agreement) framework for cloud computing to help potential NCSs to identify necessary protection mechanisms and facilitate automatic service composition. In [6], different virtual resource orchestration constraints are resumed and expressed by Attribute-Based Paradigm from NSC perspective. The other trend which is part of our work takes into consideration NSP capacities and offers in order to perform the selection. In [20], both NSP and NSC can express security requirements in SLA contract then these security requirements are transformed to OrBAC [15] policies. Li et al. [21] proposes a method to measure the similarity between security policies and suggest using the solution in SP selection process.

Most of the work in the literature define security policies negotiation based on access control negotiation. The literature is classified in 3 types of negotiations: (1) negotiation with no constraints, (2) negotiation with global constraints, (3)

negotiation with local constraints [11]. For example, [5] examines the problem of negotiating a shared access state, assuming all negotiators use the RBAC [30] policy model. Based on a mathematical framework, negotiation is modeled as a Semiring-based Constraint Satisfaction Problem (SCSP) [7]. In [35], authors argue that the guidance provided by constraints is not enough to bring practical solutions for automatic negotiation. Thus, they define an access control policy language which is based on Datalog [14] with constraints and the language can be used to define formal semantics of XACML [28]. Towards the need for human consent in organizational settings, Mehregan et al. [22] develop an extension of Relationship-Based Access Control (ReBAC) model [10] to support multiple ownership, in which a policy negotiation protocol is in place for co-owners to come up with and give consent to an access control policy. Some autors consider security policy negotiation as a process of contract establishment. For example, Li et al. [19] propose to integrate policy negotiation in contract negotiation by introducing bargaining process. An extension of the negotiation model is proposed in [29] and the model is designed for privacy policy negotiation in mobile health-Cloud environments. In our work we combine the 3 negotiation types and adapt them to negotiate SDN firewall policies in the cloud.

Our solution fills the aforementioned gaps. It meets key-functional requirements for user-centric clouds as (1) it addresses the firewall service configuration at the management layer. (2) it offers a language for firewall policies expression. (3) It supports multiple policy models in order to translate attribute-based security expression to concrete policies. (4) It selects the best SDN Firewall NSP. (5) It provides a negotiation protocol for NSC and NSP based on 3 types of negotiations. (6) it establishes a service level agreement between NSC and NSP. (7) It reinforces the contract according to SDN infrastructure configuration. (8) Then, it interprets it into OpenFlow rules and deploys them inside the network. To the best of our knowledge, there is no method in the literature that considers all these points together.

2 SDN firewall policy provisioning model

Both NSC and NSPs specify their firewall policies using our proposed expression Language. Then the Orchestrator assesses the expressions by comparing NSP's service templates with NSC's policies after receiving them. It ranks the NSPs and selects the best one which fulfills the most NSC's requirements. It starts a negotiation process with NSC in the case it is necessary. A successful negotiation generates a firewall policy agreement. This contract is derived from high-level firewall policies and sent to the chosen SDN firewall service. The latter reinforces the received policies according to its view (topology, previous security policies and other network configurations) and translates them into OpenFlow rules. Afterwards, it sends the OpenFlow rules to the SDN controllers. Each one of them deploys the received OpenFlow rules on its network elements.

2.1 Scenario Description

We introduce a use case to experiment our concept. The subjects involved in the scenario are NSC, SDN orchestrator and 3 NSPs. NSC requires an SDN Firewall service that meets its firewall policies (Requirements). Each NSP provides a type of SDN firewalls and a set of firewall policies (Obligations). The three SDN firewall services are as follows:

1. **NSP1: SDN Reactive Stateful Firewall** [37]. It forwards systematically all the packets to the stateful firewall Application over the SDN controller. The application verifies each packet using its access control table and reacts to these network events by installing the proper stateful firewall OpenFlow rules in the Network Elements. It relies on the Finite State Machine of all the connections. This service spares Network elements resources. However, it shifts the computing and memory loads on the controller. As a result, the latter became vulnerable to some DDoS attacks.
2. **NSP2: SDN Proactive Stateful Firewall** [36]. The service is based on a white list approach. It closes all the accesses and opens only the routes to the authorized connections while tracking their states. The service pre-installs all the Stateful firewall OpenFlow rules in the Network Elements. The latter sends each time a copy of their events to the Firewall service. The proactive service protects against DDoS attacks. It delegates also the access control to the Network Elements.
3. **NSP3: Stateless SDN Firewall** [16],[31]. The service does not track the connections states and it is vulnerable to DDoS attacks. However it consumes fewer resources in the Network Elements and in the Controller comparing with the above services.

2.2 Expression of Firewall Policies

We propose an SDN firewall policy language to homogenize NSP's obligations and NSC's requirements. The proposed language is inspired from the Attribute-Based Access Control Model (ABAC) [8]. It allows expressing firewall policies based on a common template. These unification guarantees the interoperability between the Obligations and the Requirements. The grammar of our language is as follows:
Π is the set of all the firewall policies. It describes the access controls within the dynamic environment of the allocated cloud resources: $\Pi = \{\pi_1, \pi_2, ..., \pi_m\}$ where $\pi_{i_{=1..m}}$ are firewall policies.
Θ is a set of Obligations. It encompasses all the firewall policies of Π expressed by NSPs. $\Theta = \{\theta_1, \theta_2, ..., \theta_k\}$
Φ is a set of Requirements. It encompasses all the firewall policies of Π expressed by NSC. $\Phi = \{\phi_1, \phi_2, ..., \phi_j\}$
Where $\Pi = \Theta \cup \Phi$ and $\pi_{i_{=1..m}} \equiv \theta_{i_{=1..k}} \vee \phi_{i_{=1..j}}$
Each firewall policy π_i is formed by many atomic elements $\varepsilon_{i_{=1..n}}$:
$\pi_{i_{=1..m}} \equiv \varepsilon_1 \wedge \varepsilon_2 \wedge ... \wedge \varepsilon_n$

$\varepsilon_{i_{=1..n}}$ is defined by a preposition of predicates. Each predicates defines a propriety of the *element*.

Theorem 1. *$A(\varepsilon_i)$ and $B(\varepsilon_i)$ are two predicates defining ε_i proprieties. Predicates equivalence is determined by the preposition : $A(\varepsilon_i) \in \Omega, B(\varepsilon_i) \in \Psi \mid (\Psi = \Omega) \rightarrow (A(\varepsilon_i) \equiv B(\varepsilon_i))$.*

The atomic rule element $\varepsilon_{i_{=1..n}}$ is formed by the following predicates :

1. **Type:** $type(\varepsilon_i) \equiv subject(\varepsilon_i) \vee action(\varepsilon_i) \vee object(\varepsilon_i) \vee context(\varepsilon_i)$
2. **Domain:** $domain(type(\varepsilon_i)) \in \{protocol, time...\}$. Domain restricts the unit of an element.
3. **Value:** $value(type(\varepsilon_i)) \equiv variable(type(\varepsilon_i)) \vee non\text{-}variable(type(\varepsilon_i))$. *variable* has not an assigned value while *non-variable* has an already assigned value. Both *variable* and *non-variable* can be assigned by three kinds of data types:
   (a) **constant:** numeric or semantic value, ex. $value(type(\varepsilon_i)) = TCP$.
   (b) **interval**: numeric interval, ex. $value(type(\varepsilon_i)) = [8:00,\ 20:00]$
   (c) **set**: a collection of values, ex. $value(type(\varepsilon_i)) = \{15:00, 16:00\}$

   For simplification, we use x_i to present a *variable*, $x_i \equiv variable(type(\varepsilon_i))$
4. **Scope**: it defines the access to the values of a *variable*. It can be:
   (a) **Public preference**: $\mathbf{pub_{pre}}(x_i)$ a public preference *variable* is accessible as public information.
   (b) **Private preference**: $\mathbf{pri_{pre}}(x_i)$ a private preference *variable* is a local configuration that can not be disclosed.

If *context* is not specified in a policy we add a universal context element $\top$. It indicates that all the obligations for the context are acceptable.
$(context(\varepsilon_i) \equiv \top) \rightarrow ((domain(context(\varepsilon_i)) \equiv \top) \wedge (value(context(\varepsilon_i)) \equiv \top))$
Finally we write:
$\varepsilon_i \equiv type(\varepsilon_i) \wedge domain(type(\varepsilon_i)) \wedge value(type(\varepsilon_i)) \wedge (\mathbf{pub_{pre}}(x_i) \vee \mathbf{pri_{pre}}(x_i))$
When the scope is not defined: $\varepsilon_i \equiv type(\varepsilon_i) \wedge domain(type(\varepsilon_i)) \wedge value(type(\varepsilon_i))$
The firewall policies given in 2.1 using our language are defined in Table 1.

2.3 Assessment of Firewall Policies

The assessment of firewall policies is based on matching the Obligations with the Requirements in order to determine which NSPs' policies satisfies NSC's requests. This process depends on two level of relationships. Element-Element relation which relies on corresponding the predicates of the firewall policies elements. The second level (Policy-Policy relation) focuses on finding the relationships between the matched elements.

Table 1. Firewall Policy Expression for NSC, NSP1, NSP2 and NSP3

NSC	ϕ_1				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	time
	value	{NSC, NSP}	pass	{HTTP, TCP, ICMP}	[0:00,24:00]
	ϕ_2				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	connection_exc
	value	{NSC, NSP}	x_2	TCP	TCP_failed_Time >30
	Scope	-	$\mathbf{pri_{pre}}$ ({quarantine, block, alert})	-	-
	ϕ_3				
	Element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	attack_detection
	value	{NSC, NSP}	block	ICMP	DoS_detection
NSP1	θ_1				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	time
	value	NSP	x_2	x_3	x_4
	Scope	-	$\mathbf{pub_{pre}}$({pass, block})	$\mathbf{pub_{pre}}$({HTTP, TCP, ICMP})	$\top$
	θ_2				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	connection_exc
	value	NSP	x_2	x_3	x_4
	Scope	-	$\mathbf{pri_{pre}}$({block, alert})	$\mathbf{pub_{pre}}$({TCP, HTTP, SSH, ICMP})	$\top$
NSP2	θ_1 **(same policy as NSP1)**				
	θ_2 **(same policy as NSP1)**				
	θ_3				
	Element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	protocol	attack_detection
	value	NSP	block	x_3	x_4
	Scope	-	-	$\mathbf{pub_{pre}}$({HTTP, TCP,SSH, ICMP})	$\mathbf{pub_{pre}}$({Poisoning, DoS_detection})
NSP3	θ_1 **(same policy as NSP1)**				
	θ_4				
	element	ε_1	ε_2	ε_3	ε_4
	type	subject	action	object	context
	domain	organization	firewall operation	IP_address	$\top$
	value	NSP	x_2	x_3	x_4
	Scope	-	$\mathbf{pub_{pre}}$({pass, block})	$\top$	$\top$

Element-Element relation. There are five relations between the elements:

1. **inconsistent**: ($type(\varepsilon_i) \not\equiv type(\varepsilon_j)) \rightarrow (\varepsilon_i \dashv\vdash \varepsilon_j)$. If two rule elements ε_i and ε_j have not equivalent *type* predicates then they are in *inconsistent* relation denoted: $\varepsilon_i \dashv\vdash \varepsilon_j$. For example, in Table 1, $\phi_1.\varepsilon_1 \dashv\vdash \theta_1.\varepsilon_2$ because $subject(\phi_1.\varepsilon_1) \not\equiv action(\theta_1.\varepsilon_2)$

   Theorem 2. *Not equivalence of type is defined as follows:*
   $type(\varepsilon_i) \in \Omega, type(\varepsilon_j) \in \Psi \mid ((\Omega \not\subseteq \Psi) \wedge (\Psi \not\subseteq \Omega)) \rightarrow (type(\varepsilon_i) \not\equiv type(\varepsilon_j))$

2. **comparable**: $((type(\varepsilon_i) \equiv type(\varepsilon_j)) \wedge (domain(\varepsilon_i) \cong domain(\varepsilon_j))) \rightarrow (\varepsilon_i \sim \varepsilon_j)$. If two rule elements ε_i and ε_j have equivalent *type* predicates and their domain predicates are in congruence, then they are in *comparable* relation. It is denoted with $\varepsilon_i \sim \varepsilon_j$. For example, in Table 1, $\phi_1.\varepsilon_2 \sim \theta_1.\varepsilon_2$ because their *subject* predicates are equivalent and their *domain* predicates are congruent.

   Theorem 3. *Domain congruence is defined as follows:*
   $domain(\varepsilon_i) \in \Omega, domain(\varepsilon_j) \in \Psi \mid ((\Omega \subseteq \Psi) \vee (\Psi \subseteq \Omega)) \rightarrow (domain(\varepsilon_i) \cong domain(\varepsilon_j))$

3. **equal**: $((\varepsilon_i \sim \varepsilon_j) \wedge (value(type(\varepsilon_i)) \cong value(type(\varepsilon_j)))) \rightarrow (\varepsilon_i = \varepsilon_j)$. If two rule elements ε_i and ε_j are *comparable* and their values predicates are in congruence, then they are in *equal* relation denoted with $\varepsilon_i = \varepsilon_j$. For example, in Table 1, $\phi_2.\varepsilon_3 = \theta_3.\varepsilon_3$ because both elements are *comparable* and their *value* predicates are congruent ($\{TCP\} \subseteq \{HTTP, TCP, SSH, ICMP\}$).

   Theorem 4. *Value congruence is defined as follows:*
   $value(type(\varepsilon_i)) \in \Omega, value(type(\varepsilon_j)) \in \Psi \mid$
   $((\Omega \subseteq \Psi) \vee (\Psi \subseteq \Omega)) \rightarrow (value(type(\varepsilon_i)) \cong value(type(\varepsilon_j)))$

4. **unequal**: $((\varepsilon_i \sim \varepsilon_j) \wedge (value(type(\varepsilon_i)) \not\equiv value(type(\varepsilon_j)))) \rightarrow (\varepsilon_i \neq \varepsilon_j)$. If two rule elements ε_i and ε_j are *comparable* but do not have equivalent value, they are in *unequal* relation denoted with $\varepsilon_i \neq \varepsilon_j$. For example, in Table 1, $\phi_1.\varepsilon_2 \neq \theta_3.\varepsilon_2$ because both are comparable however they have not equivalent values ($pass \not\equiv block$).

   Theorem 5. *Not equivalence of value is defined as follows:*
   $type(\varepsilon_i) \in \Omega, type(\varepsilon_j) \in \Psi \mid ((\Omega \not\subseteq \Psi) \wedge (\Psi \not\subseteq \Omega)) \rightarrow (value(type(\varepsilon_i)) \not\equiv value(type(\varepsilon_j)))$

5. **incomparable**: $((type(\varepsilon_i) \cong type(\varepsilon_j)) \wedge (domain(\varepsilon_i) \not\equiv domain(\varepsilon_j))) \rightarrow (\varepsilon_i \nsim \varepsilon_j)$. If two rule elements ε_i and ε_j have equivalent *type* predicates and not equivalent *domain* predicate, then they have *incomparable* relation denoted with $\varepsilon_i \nsim \varepsilon_j$. For example, in Table 1, $\phi_1.\varepsilon_3 \nsim \theta_4.\varepsilon_3$ because they have congruent *type* predicates but their domains are not equivalent ($protocol \not\equiv IP\_address$).

   Theorem 6. *Congruence of type is defined as follows:*
   $type(\varepsilon_i) \in \Omega, type(\varepsilon_j) \in \Psi \mid ((\Omega \subseteq \Psi) \vee (\Psi \subseteq \Omega)) \rightarrow (type(\varepsilon_i) \cong type(\varepsilon_j))$

Policy-Policy relations. We derive from Element-Element relations, three relations between policies. These relations are as follows:

1. **match**: $(P_1 \wedge P_2 \wedge ... \wedge P_n) \rightarrow (\pi_\alpha \bowtie \pi_\beta)$
   $P_i \equiv ((\pi_\alpha.\varepsilon_i = \pi_\beta.\varepsilon_1) \vee ... \vee (\pi_\alpha.\varepsilon_i = \pi_\beta.\varepsilon_n)) \mid \pi_\alpha, \pi_\beta \in \Pi, i = 1..n$
   If any *element* of a policy α is in equal relation with another element of a policy β, then the two policies are in *match* relation denoted with $\pi_\alpha \bowtie \pi_\beta$. For example, in Table 1, $\phi_3 \bowtie \theta_3$ because $(\phi_3.\varepsilon_1 = \theta_3.\varepsilon_1) \wedge (\phi_3.\varepsilon_2 = \theta_3.\varepsilon_2) \wedge (\phi_3.\varepsilon_3 = \theta_3.\varepsilon_3) \wedge (\phi_3.\varepsilon_4 = \theta_3.\varepsilon_4)$.
2. **mismatch**: $\exists\varepsilon_i \exists\varepsilon_j (\pi_\alpha.\varepsilon_i \nsim \pi_\beta.\varepsilon_j) \rightarrow (\pi_\alpha \asymp \pi_\beta) \mid \pi_\alpha, \pi_\beta \in \Pi$. If there are at least *incomparable* elements ε_i and ε_j from two policies π_α and π_β, then the two policies have *mismatch* relation denoted with $\pi_\alpha \asymp \pi_\beta$. For example, in Table 1, $\phi_3 \asymp \theta_2$ because $\phi_3.\varepsilon_4 \nsim \theta_2.\varepsilon_4$.
3. **potential match**: $((\forall\varepsilon_i \forall\varepsilon_j (\pi_\alpha.\varepsilon_i \sim \pi_\beta.\varepsilon_j)) \wedge (\exists\varepsilon_k \exists\varepsilon_l (\pi_\alpha.\varepsilon_k \neq \pi_\beta.\varepsilon_l))) \rightarrow (\pi_\alpha \propto \pi_\beta) \mid \pi_\alpha, \pi_\beta \in \Pi$. If all the elements of the policies π_α and π_β are *comparable* but it exists at least an *unequal* relation between two of their respective elements then the two policies are in a *potential match* relation denoted $\pi_\alpha \propto \pi_\beta$. For example, in Table 1, $\phi_1 \propto \theta_1$ because $((\phi_1.\varepsilon_1 \sim \theta_1.\varepsilon_1) \wedge (\phi_1.\varepsilon_2 \sim \theta_1.\varepsilon_2) \wedge (\phi_1.\varepsilon_3 \sim \theta_1.\varepsilon_3) \wedge (\phi_1.\varepsilon_4 \sim \theta_1.\varepsilon_4)) \wedge (\phi_1.\varepsilon_1 \neq \theta_1.\varepsilon_1)$.

NSP Ranking. The orchestrator ranks each NSP based on the relations between the *Requirements* and the *Obligations* (see Section 2.3). This process enables selecting the most compliant NSP according to the algorithm 1.

2.4 Establishment of contract

Negotiation Protocol. When an agreement is not reached between the peers, the orchestrator negotiates the offers of the chosen NSP with NSC. We propose the Rule-Element Based Negotiation Protocol (RENP) in order to manage the negotiation process. Our protocol specifies for each element the next action regarding the proposed values (v_{rec}) of NSP and the local configuration (v_{loc}) of NSC. The protocol contains three types of actions:

1. **accept**: it indicates that the proposed value is agreed.
2. **refuse**: it indicate that the proposed value is aborted.
3. **propose**: It generates a counter-offer .

Table 4 in the Appendix presents the detail of RENP protocol. (vp) is the proposed variable upon negotiation. The results of the assessment and negotiation processes for the example defined in Section 2.1 are as follows. The orchestrator finds *potential match* relations between the pairs: (ϕ_1, θ_1), (ϕ_2, θ_2) and (ϕ_3, θ_3). NSP1 does not meet the firewall requirement of ϕ_3 and NSP3 does not fulfill ϕ_2 and ϕ_3. As a consequence, the resulting relations are *mismatch*. The orchestrator puts NSP2 into the ranking list.

The orchestrator conducts then policy negotiation with NSC on behalf of NSP2. It accepts the obligations θ_1 for ϕ_1 and θ_3 for ϕ_3. However, it proposes

Algorithm 1 NSP Ranking

```
rank_list is Empty {Initial ranking list is Empty}
for All NSPs do
    rank ← 0 {Default ranking value}
    matching ← True {To make sure that there are only matches for Gold ranking}
    for i=0, i≤Length(Requirement), i++ do
        for j=0, j≤Length(Obligation), j++ do
            if Match(Requirement[i], Obligation[j]) = True then
                rank ← 2
                Break
            else if Potential_Match(Requirement[i], Obligation[j]) = True then
                rank ← 1
                matching ← False
                Break
            else if Mismatch(Requirement[i], Obligation[j]) = True then
                matching ← False
            end if
        end for
        if rank = 0 then
            Break
        end if
    end for
    if (rank = 2) and (matching = True) then
        Add (rank_list, (NSP, NSP_gold)){Tag NSP as NSP gold and add it to the
        ranking list}
    else if rank = 1 then
        Add (rank_list, (NSP,NSP_silver)){Tag NSP as NSP silver and add it to the
        ranking list}
    else
        print NSP is not compliant with NSC, it will not be add to the ranking list
    end if
end for
return rank_list
```

a counter offer for ϕ_2. This case corresponds to column 5 in row 4 of table 4 because the received value (in ϕ_2) vp_{rec} : *quarantine* has no intersection with $pri_{pre_{loc}}$: $\{block, alert\}$ (in θ_2). Thus, the orchestrator chooses another value = *block* in $pri_{pre_{loc}}$ as a new proposition. After receiving the proposition, NSC accepts the new value because *block* belongs to the local private configuration of ϕ_2. Then the orchestrator establishes the contract between NSC and NSP2 (Table 2).

General Agreement. Algorithm 2 illustrates the contract building process conducted by the orchestrator. It chooses the NSP_{top} in the top of *rank_list*. The orchestrator accepts directly without negotiation NSP_{gold} and establishes contract with NSC. While for NSP_{silver}, it starts a negotiation process with NSC by executing the proposed negotiation protocol (see Table 4). It transforms *potential match* relations into *match* relations. If the negotiation fails NSP is deleted from *rank_list* and the negotiation process is re-conducted.

Algorithm 2 Establishment of contract

rank_list← **call (NSP Ranking)** {Making NSP Ranking List}
while NSP in rank_list **do**
  $Best\_NSP = NSP_{top}$ {Choose NSP_{top} from rank_list}
  if $Best\_NSP$ is NSP_{gold} **then**
    Accept ($Best\_NSP.Obligation()$) { Accept NSP_{top} offer }
    $Contract$ =**Generate_Contract** ($Best\_NSP, NSC$) {Establish contract with NSC}
    return
  else
    Negociate ($Best\_NSP.Obligation()), NSC.Requirement()$) {Start negotiation with NSC}
    $negotiation\_Result$ = **RENP** ($potential\ match$) {Execute negotiation protocol between $potential\ match$ rule pairs}
    if $negotiation\_Result = Accepted$ **then**
      $Contract$ =**Generate_Contract** (NSP, NSC) {Establish contract with NSC}
      return
    else
      Delete (NSP_{top}, rank_list) { Delete NSP_{top} from rank_list }
    end if
  end if
end while

Table 2. Final agreement between NSP and SDN Orchestrator

NSC	**Policy 1**				
	Element	ϵ_1	ϵ_2	ϵ_3	ϵ_4
	Type	subject	action	object	context
	Domain	organization	firewall operation	protocol	time
	value	NSP	pass	{HTTP, TCP, ICMP}	[0:00,24:00]
	Policy 2				
	Element	ϵ_1	ϵ_2	ϵ_3	ϵ_4
	Type	subject	action	object	context
	Domain	organization	firewall operation	protocol	connection_exc
	value	NSP	block	TCP	TCP_failed_Time>30
	Policy 3				
	Element	ϵ_1	ϵ_2	ϵ_3	ϵ_4
	Type	subject	action	object	context
	Domain	organization	firewall operation	protocol	attack_detection
	value	NSP	block	ICMP	DoS_detection

2.5 Enforcement of Security Policy

Policy Transformation. SDN NSP contains two levels of policy abstraction: (1) a service level abstraction which defines the business logic. This high level is expressed by administrators and tenants. (2) An OpenFlow level which interprets the high-level abstraction into infrastructure specific rules. The abstraction at the service level hides the details of the network configuration and service deployment. It simplifies the expression of the service policies. While the OpenFlow level ensures deploying the policies into the network elements according to the network state.

The orchestrator sends the high level policies to SDN Firewall Applications. Each one interprets the high level policies into OpenFlow rules and sends them to the controller.

The interpretation process is based on mapping the elements of the high level policy model with OpenFlow elements. A high level policy can be interpreted to more than one OpenFlow rule.

OpenFlow is based on flow rules. Mainly, it structures policies into 6 parts. (1) OF.Type can be *Flow ADD rules*, *Flow MODIFY rules* and *Flow DELETE rules*. (2) $Matching\ Fields$ define the characteristics of the traffic. They describe the header of a packet in order to identify network flows. (3) *Actions* specify the operations on the matched. These actions can be *Drop* traffic, $Forward\ to\ controller$, $Forward\ to\ Port$. (4) $Timers$ indicate the lifetime of the rule ($Hard\_Timeout$) or the ejection time if the rule is not matched for a time interval ($Idle\_Timeout$). (5) $Metadata$ can be used to save any extra information. (6) $Counters$ allow to specify rules based on traffic statistics. Table 3 shows the interpretation of the final agreement (table 2) into OF rules. We applied the following mappings.

1. *Object* corresponds to $OF\ Matching\ Fields$. It is the first element which is mapped to its OF counterparts. The interpretation of the object element will generate at least an OF rules for each object value.
2. *Action* of the high level policy corresponds to OF $Action\ Field$. OpenFlow offers also the possibility to express many actions ($ACTION\_List$) and to associate them to the same OF rule. The orchestrator verifies that there is no contradiction between actions (for example: *block* and *allow*) in the same policy. For example, *block* corresponds to the OF action: $DROP$.
3. $Context$ element is mapped to OF components such as $TIME\_OUTs$ and OF $Counters$ but also to firewall specific functions. The interpretation to firewall function triggers a condition in order to execute the OF rule of the $Context$. For example, the context: $TCP\_\_Failed\_Time > 30$ in $Policy2$ triggers TCP connection counting function and when it exceeds 30 connections it installs the corresponding rule for *policy*2.
4. NSP is mapped to the topology of the service provider. For each link between the nodes, the interpretation module generates the corresponding OF rules taking the 3 aforementioned mappings. The OF Matching fields that correspond to the topology are at least $IP_{src}, IP_{dst}, PORT_{src}$, and $PORT_{dst}$. If the topology is not provided, the Firewall Application installs the OF rules without specifying the topology matching fields.
5. The default type of OF rule is ADD. The firewall application verifies firstly that the rule is not a duplicate of a previous interpreted rule by comparing both matching parts. If only the contexts are different, then the OF rule type is set to $MODIFY$. If the firewall application receives from the Controller an error upon sending a $MODIFY$ rule, the firewall application changes the $MODIFY$ rule to ADD rule and re-sends it to the controller.

Policy Deployment. The controller opens a secure channel with the data plane devices (network elements) and communicates by exchanging OpenFlow

Table 3. Interpretation of the Final Agreement into OpenFlow Rules

	OF Type	**Matching Field**	**Action**	**Timer**	**Firewall Function**
Policy 1	ADD	ETH_Type=2048 IP_Proto=6 DST_P=80	Forward Controller	Idle_Timeout=0 Hard _Timeout=0	
Policy 1	ADD	ETH_Type=2048 IP_Proto=6	Forward Controller	Idle_Timeout=0 Hard _Timeout=0	
Policy 1	ADD	ETH_Type=2048 IP_Proto=1	Forward Controller	Idle_Timeout=0 Hard _Timeout=0	
Policy 2	MODIFY	ETH_Type=2048 IP_Proto=6	DROP		TCP_Count(30)
Policy 3	MODIFY	ETH_Type=2048 IP_Proto=1	DROP		SNORT.ALERT =DDOS

messages. Upon receiving OF rules from SDN firewall, the controller sends them to the corresponding data plane devices which then install the rules. The rules are parsed and their elements are saved in the Flow tables of the data plane devices. At this level, the OF rules become Flow entries in the data plane devices. When the data plane devices receive network packets, they parse their headers and match the contents with all the matching fields of each flow entry. Once a matching is found, the corresponding action is executed on the packet. If a match is not found, the data plane device drops the packet or sends it to the controller.

3 Evaluation

We implement the proposed solution using Python programming language. The Framework has an orchestrator layer which integrates the firewall provisioning model. In addition, it has a stateful SDN firewall application that executes the security policies and the finite state machines of stateful network protocols.

We deploy our solution in the B-Secure platform which is a cloud environment to test the performance of SDN security solutions. It consists of a central machine (16 Gb of RAM and Intel i7 processors), a data plane device machine (16 Gb of RAM, Intel i7 processors, 6 physical network interfaces of 1 Gb/s) connected to the central machine, and two machines (4 Gb of RAM and Intel i3 processors) connected to the data plane device.

We install the orchestrator, the firewall application and the SDN controller RYU [24] in the central machine. In the data plane device machine, we run the OpenVswitch (OVS) [26],[27]. It is a virtual switch framework widely used in both industry and research. The physical network link between the controller and the OVS is 1 Gb/s.

In the central machine, we deploy NSC and NSP2 of the use case (see Section 2.1). The orchestrator generates the contract and sends the high level policies to

the SDN firewall Application. Then the latter interprets the policies to OpenFlow rules and asks RYU controller to install them on OVS. We vary the number of NSC's *Requirements* from 1 to 2500 policies. We measure the following performance metrics:

1. *Policy Processing Time (PPT)* is the time that the orchestrator needs to process the policy expression.
2. *Orchestrator Processing Time (OPT)* is the total time taken by the orchestrator from the first policy expression to sending the last policy.
3. *Firewall Application Processing Time (FAPT)* is the total time taken by the firewall application to process the policies.
4. *Controller Processing Time (CPT)* is the total time taken by the controller to send all the OpenFlow rules.
5. *Infrastructure Processing Time (IPT)* is the total time that the infrastructure (Controller-OVS link and OVS) needs to install all the OF rules.
6. *Policy Processing Total Time (PPTT)* is the total policy provisioning time.
7. *Orchestrator Setup Rate (OSR)* is the speed of the orchestrator:

$$OSR = OPT/Number\,of\,Policies \tag{1}$$

8. *Firewall Setup Rate (FSR)* is the speed of the firewall Application:

$$FSR = FAPT/Number\,of\,Policies \tag{2}$$

9. *Controller Setup Rate (CSR)* is the speed of the Controller:

$$CSR = CPT/Number\,of\,Openflow\ Rules \tag{3}$$

10. *Infrastructure Setup Rate (ISR)* is the speed of the Infrastructure:

$$ISR = IPT/Number\,of\,Openflow\ Rules \tag{4}$$

Figure 1 and Figure 2 display the different measured processing times during the experiment. The processing times in all the figures increase with the rise of rule number. In Figure 1, we observe that PPT increases slowly with a starting value of $0.00236s$ for 10 policies to a maximal value of $0.6421s$ for 2500 policies. In addition, PPT's values are the lowest among all processing times. $FAPT$ is slightly higher than CPT. Moreover, OPT is slower than both $FAPT$ and CPT. For example, the values for 2500 policies are : $2.700s$, $2.254s$ and $2.141s$. However, the largest processing times are those of IPT as shown in Figure 2 ($(10, 0.0034s)$ and $(2500, 11.025s)$).

The reasons are the amount and nature of the processing that each layer performs. The orchestrator runs many processes in order to generates the final agreement. The firewall application performs the interpretation to OF rules and the controller deploys the generated OF rules in the network element. The infrastructure is impacted by the performance of OVS and the data link with the controller. Our solution accumulates a processing time of $7.96s$ with 2500 rules, while the infrastructure processing time is 1.5 times higher $(2500, 11.02s)$.

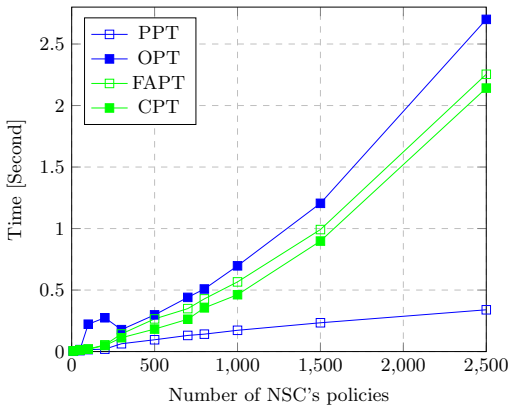

Fig. 1. Policy processing Times

Around 50% (in Average) of $PPTT$ is taken by the infrastructure to deploy the rules. For example, for 2500 rules, it takes $18.12s$ from the time of releasing the initial policy request in the orchestrator to the time of deployment of the final OF rule in the network element. 60% of this time is taken by the infrastructure alone (see Figure 2).

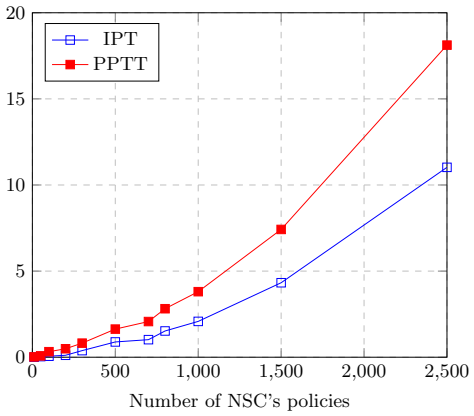

Fig. 2. Policy Deployment Times

Figure 3 displays the different policy setup rates in the infrastructure. We observe 3 different states. In the first state, CSR, FSR, OSR and ISR increase to reach their top values respectively $(100, 5186)$, $(100, 4838)$, $(50, 3150)$, $(10, 2941)$. In the second stage, all the rates decrease rapidly. The diminution is linear for CSR, FSR and OSR while fluctuating for ISR. In the third stage, we observe

that all the rates reduce with the increase of the number of policies. The rates of our solution reach a value around 1000 $Policies/s$ at 2500 while ISR continues to hold lower values. This observation comforts the previous results. ISR low rates are caused by the load on the link Controller-OVS. We observe that the orchestrator has lower performance than the firewall application. This observation consolidates the explanations provided previously. Our solution has good performances with around 1000 policies/s. In practice, the number of firewall rules depends on the size of the topology and the granularity of each rule. Furthermore, policies changes do not need the repetition of all the process because OpenFlow enables to update directly the installed rules.

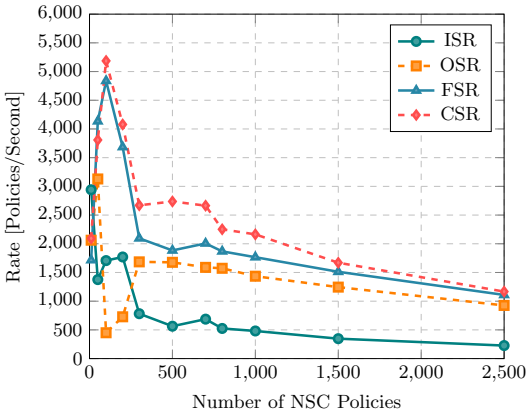

Fig. 3. Solution's Policy Rates

4 Conclusion and Perspectives

We propose a solution to express firewall policies, assess NSC's requirements with NSPs' obligations, select the best NSP candidate, negotiate and agree on a common policy contract then deploy the agreement in a SDN cloud platform. We integrate and deploy the solution in an existing SDN firewall framework. Moreover, we evaluate its performance and scalability. The evaluation shows promising results with a rate of 1000 deployed policies/s.

Our framework brings many advantages. It offers interoperability between different NSCs and NSPs through a unified language that simplifies administrator's tasks. It abstracts the complexity of the network by hiding the infrastructure details. In addition, it automatizes firewall policies orchestration.

In order to improve our solution, we plan to include in the process new elements such as quality of service, pricing and NSP's reputation. This improvement will resolve specific cases in our ranking algorithm. For example, the empty rank_list or multiple matching and potential matching NSPs.

Acknowledgement

The work of Nora Cuppens and Frédéric Cuppens reported in this paper has been partially carried out in the SUPERCLOUD project, funded by the European Unions Horizon 2020 research and innovation programme under grant N643964.

References

1. Adrian Lara, A.K., Ramamurthy, B.: Network innovation using openflow: A survey. IEEE COMMUNICATIONS SURVEYS & TUTORIALS 16(1), 493–511 (2014)
2. Batista, B., Fernandez, M.: Ponderflow: A policy specification language for openflow networks. In: The Thirteenth International Conference on Networks. pp. 204–209 (2014)
3. Ben-Itzhak, Y., Barabash, K., Cohen, R., Levin, A., Raichstein, E.: Enforsdn: Network policies enforcement with sdn. In: Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on. pp. 80–88. IEEE (2015)
4. Bernsmed, K., Jaatun, M.G., Undheim, A.: Security in service level agreements for cloud computing. In: CLOSER. pp. 636–642 (2011)
5. Bharadwaj, V.G., Baras, J.S.: Towards automated negotiation of access control policies. In: Policy. pp. 111–119 (2003)
6. Bijon, K., Krishnan, R., Sandhu, R.: Virtual resource orchestration constraints in cloud infrastructure as a service. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy. pp. 183–194. ACM (2015)
7. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., Fargier, H.: Semiring-based csps and valued csps: Frameworks, properties, and comparison. Constraints 4(3), 199–240 (1999)
8. Chernov, D.V.: Attribute based access control models. Prikladnaya Diskretnaya Matematika. Supplement pp. 79–82 (2012)
9. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: Policies for Distributed Systems and Networks, pp. 18–38. Springer (2001)
10. Fong, P.W.: Relationship-based access control: protection model and policy language. In: Proceedings of the first ACM conference on Data and application security and privacy. pp. 191–202. ACM (2011)
11. Gligor, V.D., Khurana, H., Koleva, R.K., Bharadwaj, V.G., Baras, J.S.: On the negotiation of access control policies. In: International Workshop on Security Protocols. pp. 188–201. Springer (2001)
12. Hegr, T., Bohac, L., Uhlir, V., Chlumsky, P.: Openflow deployment and concept analysis. Advances in Electrical and Electronic Engineering 11(5), 327 (2013)
13. Hu, H., Han, W., Ahn, G.J., Zhao, Z.: Flowguard: building robust firewalls for software-defined networks. In: Proceedings of the third workshop on Hot topics in software defined networking. pp. 97–102. ACM (2014)
14. Huang, S.S., Green, T.J., Loo, B.T.: Datalog and emerging applications: an interactive tutorial. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. pp. 1213–1216. ACM (2011)
15. Kalam, A.A.E., Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. In: Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on. pp. 120–131. IEEE (2003)

16. Kaur, K., Kaur, S., Gupta, V.: Software defined networking based routing firewall. In: Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on. pp. 267–269. IEEE (2016)
17. Lara, A., Ramamurthy, B.: Opensec: Policy-based security using software-defined networking. IEEE Transactions on Network and Service Management 13(1), 30–42 (2016)
18. Leite, A.F., Alves, V., Rodrigues, G.N., Tadonki, C., Eisenbeis, C., de Melo, A.C.M.A.: Automating resource selection and configuration in inter-clouds through a software product line method. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 726–733. IEEE (2015)
19. Li, Y., Cuppens-Boulahia, N., Crom, J.M., Cuppens, F., Frey, V.: Reaching agreement in security policy negotiation. In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 98–105. IEEE (2014)
20. Li, Y., Cuppens-Boulahia, N., Crom, J.M., Cuppens, F., Frey, V.: Expression and enforcement of security policy for virtual resource allocation in iaas cloud. In: IFIP International Information Security and Privacy Conference. pp. 105–118. Springer (2016)
21. Li, Y., Cuppens-Boulahia, N., Crom, J.M., Cuppens, F., Frey, V., Ji, X.: Similarity measure for security policies in service provider selection. In: International Conference on Information Systems Security. pp. 227–242. Springer (2015)
22. Mehregan, P., Fong, P.W.: Policy negotiation for co-owned resources in relationship-based access control. In: Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies. pp. 125–136. ACM (2016)
23. Nathani, A., Chaudhary, S., Somani, G.: Policy based resource allocation in iaas cloud. Future Generation Computer Systems 28(1), 94–103 (2012)
24. NTT: Component-based software defined networking framework (2017), `www.osrg.github.io/ryu/`
25. ONF: Openflow switch specification (December 2014)
26. Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., Shenker, S.: Extending networking into the virtualization layer. In: Hotnets (2009)
27. Pfaff, B., Pettit, J., Koponen, T., Jackson, E.J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., et al.: The design and implementation of open vswitch. In: NSDI. pp. 117–130 (2015)
28. Rissanen, E.: extensible access control markup language (xacml) version 3.0 (committe specification 01). Tech. rep., Technical report, OASIS, http://docs. oasisopen. org/xacml/3.0/xacml-3.0-core-spec-cd-03-en. pdf (2010)
29. Sadki, S., El Bakkali, H.: An approach for privacy policies negotiation in mobile health-cloud environments. In: Cloud Technologies and Applications (CloudTech), 2015 International Conference on. pp. 1–6. IEEE (2015)
30. Sandhu, R.S., Coynek, E.J., Feinsteink, H.L., Youmank, C.E.: Role-based access control models yz. IEEE computer 29(2), 38–47 (1996)
31. Satasiya, D., et al.: Analysis of software defined network firewall (sdf). In: Wireless Communications, Signal Processing and Networking (WiSPNET), International Conference on. pp. 228–231. IEEE (2016)
32. Shin, S., Gu, G.: Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In: 2012 20th IEEE international conference on network protocols (ICNP). pp. 1–6. IEEE (2012)

33. Shin, S., Porras, P.A., Yegneswaran, V., Fong, M.W., Gu, G., Tyson, M.: Fresco: Modular composable security services for software-defined networks. In: NDSS (2013)
34. Tang, Y., Cheng, G., Xu, Z., Chen, F., Elmansor, K., Wu, Y.: Automatic belief network modeling via policy inference for sdn fault localization. Journal of Internet Services and Applications 7(1), 1 (2016)
35. Xue, W., Huai, J., Liu, Y.: Access control policy negotiation for remote hot-deployed grid services. In: First International Conference on e-Science and Grid Computing (e-Science'05). pp. 9–pp. IEEE (2005)
36. Zerkane, S., Espes, D., Le Parc, P., Cuppens, F.: A proactive stateful firewall for software defined networking. In: Risks and Security of Internet and Systems - 11th International Conference, CRiSIS 2016, Roscoff, France, September 5-7, 2016, Revised Selected Papers. pp. 123–138 (2016)
37. Zerkane, S., Espes, D., Le Parc, P., Cuppens, F.: Software defined networking reactive stateful firewall. In: ICT Systems Security and Privacy Protection - 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30 - June 1, 2016, Proceedings. pp. 119–132 (2016)

A RENP Protocol

Table 4. RENP protocol

v_{loc} / v_{rec}	non variable	variable $\mathbf{pub_{pre}}$	variable $\mathbf{pri_{pre}}$	proposed value (**vp**)
non variable	$(v_{rec} = v_{loc})$ $\rightarrow accept(v_{rec})$ $(v_{rec} \neq v_{loc})$ $\rightarrow refuse$	$(\{v_{loc}\} \subseteq \{pub_{pre_{rec}}\})$ $\rightarrow propose(v_{loc})$ $(\{v_{loc}\} \nsubseteq \{pub_{pre_{rec}}\})$ $\rightarrow refuse$	$propose(v_{loc})$	-
variable $\mathbf{pub_{pre}}$	$(\{v_{rec}\} \subseteq \{pub_{pre_{loc}}\})$ $\rightarrow accept(v_{rec})$ $(\{v_{rec}\} \nsubseteq \{pub_{pre_{loc}}\})$ $\rightarrow refuse$	$((\{pub_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) \neq \emptyset)$ $\rightarrow propose(x)$ $x = (\{pub_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\})$ $((\{pub_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) = \emptyset)$ $\rightarrow refuse$	$propose(x)$ $x = pub_{pre_{loc}}$	$(\{vp_{rec}\} \subseteq \{pub_{pre_{loc}}\})$ $\rightarrow accept(vp_{rec})$ $(\{vp_{rec}\} \nsubseteq \{pub_{pre_{loc}}\})$ $\rightarrow refuse$
variable $\mathbf{pri_{pre}}$	$(\{v_{rec}\} \subseteq \{pri_{pre_{loc}}\})$ $\rightarrow accept(v_{rec})$ $(\{v_{rec}\} \nsubseteq \{pri_{pre_{loc}}\})$ $\rightarrow refuse$	$((\{pri_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) \neq \emptyset)$ $\rightarrow propose(x)$ $x = (\{pri_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\})$ $((\{pri_{pre_{loc}}\} \cap \{pub_{pre_{rec}}\}) = \emptyset)$ $\rightarrow refuse$	$propose(x)$ $x \in \{pri_{pre_{loc}}\}$	$(\{vp_{rec}\} \subseteq \{pri_{pre_{loc}}\})$ $\rightarrow accept(vp_{rec})$ $((\{vp_{rec}\} \cap \{pri_{pre_{loc}}\}) = \emptyset)$ $\wedge \neg negotiate)$ $\rightarrow refuse$ $((\{vp_{rec}\} \cap \{pri_{pre_{loc}}\}) = \emptyset)$ $\wedge negotiate)$ $\rightarrow propose(x)$ $x \in \{pri_{pre_{loc}}\}$ $((\{vp_{rec}\} \cap \{pri_{pre_{loc}}\}) \neq \emptyset)$ $\wedge(\{vp_{rec}\} \nsubseteq \{pri_{pre_{loc}}\})$ $\wedge negotiate)$ $\rightarrow propose(x)$ $x \in (\{vp_{rec}\} \cap \{pri_{pre_{loc}}\})$