Filter Allocation Using Iterative ECNP

Jan Tožička, Štěpán Urban, Magdalena Prokopová and Michal Pěchouček

Gerstner Laboratory
Department of Cybernetics, Czech Technical University
Technická 2, Prague, 166 27, Czech Republic
{tozicka|urban|prokopova|pechouc}@labe.felk.cvut.cz

Abstract. Network devices can filter traffic in order to protect end-user computers against network worms and other threats. Since these devices have very limited memories and cannot deploy filters against every known worm, the traffic can be forwarded to other device during so called *filter delegation*. In this contribution we present two negotiation based algorithms looking for a good filter delegation solution. We formally describe this *filter allocation problem in a network* dealing with distribution of filters among agents so that several constraints are fulfilled and we extend this problem to fit a real world task. We show that both the basic problem and its extension are NP-complete. Both algorithms solving this problem are experimentally evaluated on a realistic network simulation.

1 Introduction

As data communication networks become more complex and its security more crucial then it ever was, new ways to monitor and protect them must be investigated. The software agent technology can be applied to many problems spaces from public and private to military networks on the battlefield.

Software agents seem to be a natural fit for this type of environment, they are small, robust, mobile, and have the ability to analyze and adapt to their local environment on the fly. This can also be very useful in networks with communication lags and dropouts, such as overloaded networks and wireless (Ad-Hoc) networks where the network structure can change.

Intrusion detection is an area where the agent technology can be applied. Agents that contain mobile and RC characteristics would provide a dynamic capability to discover and protect against emerging cyber threats. Knowledge sharing among agents would enable dynamic reconfiguration of the agent's capabilities and allow for distributed processing. The traditional method is to have an appliance, or application running on a server, then attack signatures would need to be put in by hand or downloaded and installed in order to add this new functionality. While the agents will still rely on similar approach and should be able to efficiently react to and contain known threats, they shall be able to use AI techniques to identify new threats similar to the known ones or emphasize irregular operations and cooperatively adapt using advanced negotiation techniques.

1.1 $\mathcal{A}$-net Network Simulation

In our work we focus on the domain of network security and reaction to intrusions. During our work we developed an agent simulation of computer network *A-net* (see Figure 1).

This agent-based network simulation provides us an easy way to deploy and test different network protection mechanisms. Each network device is represented by fully autonomous agent. The base layer of this simulation offers all common network devices, end user systems and the traffic between them. In this layer actual flow payload is not generated by the application itself, but rather one of the several hundreds of example packets is used instead. By having the large pool of example packets we can assume that this approximation is close enough to the reality. The data we are using were gathered from real-world network traffic outgoing and incoming to one of our host machines.

Additionally, we added viruses – worms that spread in the network and attack vulnerable systems – and DDoS attacks – group of bots generate huge amount of traffic targeted to one server. Using modeling of the network traffic, our intrusion detection system identifies these intrusions and creates descriptions of the malicious traffic that should be filtered out of the network, we refer to these as *filters*.

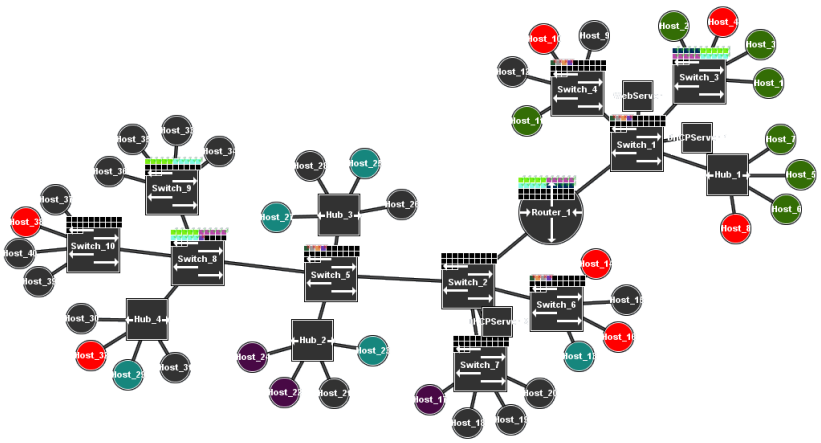

Fig. 1. $\mathcal{A}$-net network simulation.

1.2 Network Flow Filters

Each network device, such as *router, switch, hub*, is equipped with a field-programmable gate array (FPGA) and a secondary memory. The gate array

is very limited and it can contain only few filters. Nevertheless it is able to process a heavy traffic in real time (GBs per second) and filter out such a traffic that corresponds to the filters. On the other hand the secondary memory is big enough to contain specification of all worms ever created but it is very slow to be used by CPU to filter the traffic in real time.

Under these settings we try to protect all vulnerable hosts and to decrease the amount of malicious traffic in the network. In Figure 1 you can see that each network device (switches and router) have boxes which shows us deployed filters in our visualization. This box can be empty (no filter) or colored by color related to filtered traffic. Colored host means that this computer has been infected by a worm.

1.3 Filter Allocation Problem

In this article we focus on the distribution of *filters* in the network. The filters are distributed over a network so that they cover all paths between vulnerable hosts. This problem is formally described as *filter allocation problem in a network (FAPN)* and we show that it is NP-complete in the Section 4. To be able to solve real world problem we have extended the FAPN to FAPN-E which is *distributed*, *on-line*, *unbounded* and with *incomplete knowledge*. If an network device cannot deploy some of the filters other device covering the same path can deploy the filter with possibly worse utility function or the filter can be delegated to other netowrk device. In this article we present and compare two negotiation based algorithms looking for suitable filter delegation.

The rest of the paper is organized as follows. Firstly, we introduce work related to our problem in the Section 2. The Section 3 presents two algorithms searching for filter delegation. Firstly it presents ECNP based greedy algorithm in the Section 3.2 which is further extended to second algorithm described in the Section 3.3. We formally describe and analyze the complexity of FAPN problem and its extension FAPN-E in the Section 4. Both algorithms are experimentally compared in the Section 5 and we conclude this contribution in the Section 6

2 Related work

Balanced allocation of the monitoring process within the network as well as an efficient placement of the intrusion response processes need to be decided and reconfigured locally, in a peer-to-peer interaction among the network components. That is valid because there is a desire to limit centralized decision making processes and centralized collection of data in the network.

2.1 Service Oriented Architectures

Our domain is partially similar to service oriented architectures [3]. Service oriented architectures allow to create application distributed over different enterprizes. In some cases the services require local resources for their functionality

while in other cases the services can be moved to other devices and used remotely. Our filters can be viewed as services. Nevertheless the relation of service consumer-provider is not well defined in the network domain but in the filter delegation where one agent explicitely expect other agent to do filtering on his behalf.

2.2 Task Allocation Problem

Distributed task allocation is a typical problem that is solved in its different variations in research communities (e.g. [10]).

The distributed task allocation algorithms are based on different auctioning approaches (e.g. English, Vickery, Dutch, Seal-bid, All-pay [6]), each having different properties in different environments. The most widely used approaches to distributed task allocation are based on the CNP (*contract-net-protocol*) [9] (based on single round Seal-bid auction), iterated CNP, its $OSCM$-CNP optimality improvements [7] and other combinatorial auctions. These techniques have been successfully used in a number of network oriented applications. [8] focuses on similar problem to our case, the agents have limited resources and preferences and thus they have to coordinate the task distribution. The tasks do not cost anything to be promised but their consume resources when they are performed. The situation with the filters is reverse: filter deployment consumes expensive memory but its execution is free.

The problem of distributing filters through out the network is similar to the problem of resource allocation and task distribution among autonomous agents defined in [1] where a set of agents share a common resource and an agreement is sought where all agents will be able to use this resource to fulfill their goals.

Similarly, in our approach each agent contributes to the negotiation task by several inputs:

- *filters* – Each agent has its list of prioritized filters it wants to deploy.
- *resources* – Each agent taking part in the negotiation also offers some resources for filtering, either to be shared or to be used by itself
- *capabilities* – Some agents can re-delegate their tasks to different agents, but others can not.
- *strategy* – The strategy can be defined by specifying how much traffic we allow to redirect and thus increase the network traffic

Based on the criteria for evaluating negotiation protocols presented in [1] we are looking for an algorithm that is (i) *distributed*, i.e. without any central point, (ii) *simple*, i.e. the negotiation consumes reasonable amount of resources. and (iii) symmetric, i.e. all agents should be treated in the same way.

[2] describes the similar problem of task distribution, where the problem scenario has following elements

- *manager* – Agent who owns the task is referred to as manager

– *contractors* – Agents willing to cooperate with manager are called contractors
– *task* – Each task has specified benefits and resources needed
– *resources* – Resources are distributed among agents

Agents are organized into social network according to who wants to cooperate with who and they cooperate only with their neighbors. The manager starts negotiation by offering the most efficient (ratio between benefits and required resources) task to its contractors and agents choose out of all tasks the most efficient one and make a bid.

As opposed to usual approach for task distribution in our domain we need different approach to the use of the resources. In the most common scenario resources can be used to satisfy only one task, on the other hand in our case since the resources are assigned to perform certain type of task, they can be used by large number of agents at the same time[1], but only for this purpose. Therefore some adjustments to existing algorithms are necessary.

3 Filter Allocation and Delegation in $\mathcal{A}$-net

Each agent desires to fulfill all necessary filtering tasks for the lowest cost possible but its resources are limited, therefore conflict of interests can arises. Thus in order to satisfy its needs agent has to reach an agreement with other agents. In [5], we have designed an algorithm that distributes a newly created filter through the network covering all vulnerable pairs of hosts and servers – i.e. satisfying the *completeness* condition (if such a distribution exists). Moreover this algorithm uses minimal amount of resources.

Once a new filter is introduced into the network the agent that received this filter sends it to its neighbor agents. Using the same technique the filter is distributed through the network. When the filter arrives to the host machine it replies whether it is vulnerable againts described threat. Agents collect these replies and based on them decide whether to deploy the filter and how to reply to the agent that informed it about this filter.

However this distribution algorithm is optimal only for the distribution of one filter, given several filters in a row the resulting distribution is not guaranteed to be optimal. Also, this algorithm does not use the filter delegation which could allow to deploy more filters in the network. Therefore we have implemented peer to peer negotiation that further improves filter distribution.

Since, in our network simulation, we allow for a simple delegation mechanism that allows to increase number of deployed filters in the network, a network device can ask another device to filter for it and then the respective incoming traffic is forwarded traffic to other device. These delegated filters are denoted as *filter-for*, or *FF*.

In the case a filter should be placed on a device not equipped with enough available resources the process of filter delegation starts to solve the situation.

[1] Here, we consider resources needed for filter deployment only.

The idea of delegation is to find other devices with available resources where request for filter use can be forwarded.

Device can be chosen for filter delegation if

- the same filter is already deployed on the device, or
- the device has enough available resource to deploy the filter

The FAPN problem, formally described in the Section 4, is defined on an undirected graph $G = (\mathcal{A}, E)$, however we limit in our simulation to acyclic graphs only.

In this section we describe the requirements on such a solution in the Section 3.1 and then we present two solutions of filter allocation problem. Firstly, it is ECNP based greedy algorithm in the Section 3.2. We use this algorithm for evaluation of the second algorithm that uses more advanced negotiation based on an iterative modification of ECNP, described in the Section 3.3. Both algorithms are experimentally compared later in the Section 5.

3.1 Task Description

In this contribution we present and compare two algorithms finding the devices where the filters should be delegated to. ECNP protocol [4] addresses fast way how to distribute selected set of tasks. The improved iterative version of ECNP allows also to select good subset of tasks to be delegated.

When request for deploying new filter is obtained, the network device can reevaluate filters that are already deployed and according to priorities and statistics decide to remove one or more and place requested filter instead.

Apart from individual filter selection negotiation will be used also. Network devices can cooperate to find place(s) where to place filter while other devices will redirect their traffic to this place. The algorithm searching for optimal places can use following inputs and produces desired outputs:

Inputs:

- price for deploying filter
- price for redirecting traffic
  (The simple price for redirecting is given by number of nodes the traffic goes through. More elaborated measures take into account statistics – price for redirecting traffic for each filter can be multiplied by average usage of the filter.)
- priority of filter given at filter creation,
- statistics of filter usage/failures
  - number of flows filtered out / used
  - size of traffic filtered out / used
  - time since last successful filtering out / usage
- age of the filter

Outputs:

- what filters are deployed on devices
- what filters are delegated to other devices (traffic redirected)

Evaluation: Supposing we can order filters from best to worst using predefined metric (priority by itself or priority combined with statistics) we can divide the evaluation of the solution into two separate maximization/minimization tasks:

- firstly the task is maximizing number of pairs covered by filters where counting starts from the best filters
- on maximal coverage we need to minimize the increase in traffic in the network (this can be based on old statistics of filter usage and the distance how far the traffic is redirected)

By optimizing the above defined metrics the percentage of worm flows successfully filtered out should be the highest possible.

The algorithm delegating filters in the netowrk is also required to be:

- *distributed* – each agent is responsible for its netowork device and decides which filter to deploy locally
- *on-line* – filters are comming in sequence and agent need to improve current solution
- *stable* – the changes for new filter should be minimal since the cost of the reconfiguration of FPGA
- *unbounded* – there is no upper bound for the number of filters
- able to work with *incomplete knowledge* – some of the inputs are not known to the agents

3.2 Greedy Algorithm: ECNP

In the following paragraphs we describe the greedy algorithm that represents a lower bound solution of the problem.

The Algorithm 1 shows how we find new places where the filters are delegated. We are using ECNP, one of the extensions to CNP introduced in [4], where agents can bid only for parts of the offer, i.e. to choose only one or two filters out of the whole offer. Filters covered by winning bid are removed from the call-for-proposals (CfP) and the negotiation continues until all remaining filters can fit to the device itself. ECNP also introduces temporal grants and rejects, and thus the initiator of the negotiation can change its decision.

This solution is static and does not try to improve filter distribution once it is negotiated. It also delegates more filters than it is necessary. These imperfections are addressed by the improved algorithm described below.

Algorithm 1: Filter delegation greedy algorithm – ECNP.

while *Enough resource to fit all filters are available* **do**
  Send CfP for delegation of all filters that are not delegated yet to all agents.
  Wait for bids from all agents.
  Choose the best bid and confirm, reject others.
  Discard the filter locally and set delegation.
end

3.3 Improved Algorithm: Iterative ECNP

In our scenarios we are facing the challenge of *incomplete knowledge* – agents can not estimate how much traffic the filter can filter out in advance. Anytime new filter is deployed on a device, the filter evaluation becomes available after some period of time when statistics are counted. Therefore we cannot take into account the amount of filtered traffic during the initial distribution and moreover this value can change in time, thus our solution needs to be periodically checked and adjusted.

The negotiation about filter delegation can be solved one filter by one, but better results are achieved when negotiating about set of filters. We propose an algorithm that finds the best set of filters to be delegated, minimizing the unavoidable increase in network traffic.

We modified ECNP protocol to build up our knowledge about how much will the traffic increase by using filter delegation. First the value of traffic increase is estimated for each filter and the set of filters with lowest value is chosen. We use ECNP to find possible delegation places for each filter and more accurate estimations, our implementation of ECNP is however modified and no proposal is accepted. Using these more exact estimation the set of filters for delegation is recounted and ECNP is started again. The details are described in the Algorithm 2.

Algorithm 2: Iterative ECNP

Choose initial **set S** of filters for forwarding.
Set price for forwarding to initial value for all filters.
while ***set S*** *of filters to be forwarded changed* **do**
  Start non-accepting ECNP algorithm.
  Reject proposals instead of accepting them.
  Recount eventual increase in traffic based on ECNP proposals.
  Choose new **set S** of filters for forwarding.
end
Start ECNP algorithm for final **set S** and deploy filters.

We cannot use simple estimation for each filter by itself, but all estimations has to be done over a set of filters. When estimating the value of increased traffic

for a single filter the selected place for delegation is the closest agent with free resources, however this estimation can not be used for a set of filters, it is unlikely that the closest agent has enough free resources for all the filters. Therefore a set of filters is used for adjusting the estimations and the concrete distribution where to delegate is found once the set of filters for delegation is determined.

4 Formal Description of Filter Allocation Problem in Network

Let us now formally describe a *filter allocation problem in a network (FAPN)* in this section. Each agent a_i from the set of all agents $\mathcal{A} = \{a_1, \ldots, a_n\}$ has limited resources r_i available to provide some of the filters. Let $\mathcal{F} = \{f_1, \ldots, f_m\}$ be a set of filters. Each filter $f_j \in \mathcal{F}$ is defined by a tuple $< u(f_j, a_1), \ldots, u(f_j, a_n), r(f_j) >$, where $u(f_j, a_i)$ is the utility for deploying the filter f_j by an agent a_i and $r(f_j)$ is a number of resources needed to deploy the filter.

In addition there are constraints where the filters are needed. These constraints are defined on a *network* represented by an undirected graph $G = (\mathcal{A}, E)$. Each filter f_j determines a subset of paths in the G graph $\mathcal{P}_j \subseteq \text{Paths}(\mathcal{A})$, which needs to be *covered* by the filter f_j. Paths $\mathcal{P}_j$ represent paths between all pairs of vulnerable hosts in the case of worm filters or all path leading form bot net to the server they are attacking

The *distribution* of filters between agents is defined by the function $\mathcal{D} : \mathcal{F} \mapsto \mathcal{A}$. A distribution is *valid* if it satisfies following properties:

correctness : Each agent $a_i \in \mathcal{A}$ does not deploy more filters then it has resources for: $\sum_{f_j \in \mathcal{F}: \mathcal{D}(f_j) = a_i} r(f_j) < r_i$

completeness : For each filter f_j all paths $\mathcal{P}_j$ are covered.

We suppose that such a valid distribution exists for each instance of FAPN.

Under these settings the task is to maximize the overall *utility* $\mathcal{U}$:

$$\mathcal{U} = \sum_{f_j \in \mathcal{F}} u(f_j, d(f_j))$$

Problems similar tasks to FAPN are often NP-complete (e.p. task allocation problem).

Theorem 1. *For an instance of a filter allocation problem in a network, as defined in the Section 4, and a real number k the problem to decide whether distribution $\mathcal{D}$ with utility higher than k exists is NP-complete.*

Proof. Firstly, let us show that FAPN is NP problem. Having an instance of the problem, real number k and a solution $\mathcal{D}$ we can check in a polynomial time whether it is valid distribution and whether its utility is greater than k.

We use knapsack problem (KSP) to show that FAPN is NP-hard, i.e. FAPN $\leq_p$ KSP. An instance of KSP contains n items, where an item i has value v_i and size f_i, and a size of the bag c. We look for a subset of items $S \subseteq \{1, \ldots, n\}$ that

maximizes $\sum_{i \in S} v_i$ while fulfilling the constraint $\sum_{i \in S} s_i \leq c$. This instance can be transformed into FAPN with two agents $\{a_1, a_2\} : r_1 = c, r_2 = \sum_1^n s_i$ and n filters $\{f_1, \ldots, f_n\} : f_i =< \{u(f_i, a_1) = v_i, u(f_i, a_2) = 0\}, r(f_i) = s_i >$. A network connects both agents and they are present in all the sets $\mathcal{A}_i = \{a_1, a_2\}$.

A valid solution of this instance of FAPN problem can be easily transformed to a solution of original KSP problem. All items represented by filters deployed by the agent a_1 represent a solution S to KSP problem.

Variation of FAPN: FAPN-E. In this contribution we do not focus directly on presented FAPN problem but its variation will be considered instead. Let us change the FAPN problem in the following ways:

distribution – each agent is responsible for its filters

incomplete knowledge – non of the agents knows values of utility functions in advance but it can approximate the utilities of deployed filters.

on-line task – the whole set of filters is not known in advance but filters appear to the agents in sequence and the agents try to keep as good distribution as possible

unlimited size of $\mathcal{F}$ – the set of filters is increasing with the time and its size is unlimited. It means that after some time there is no valid distribution of FAPN task since the completeness property cannot be fulfilled. This case is described bellow in more detail.

The ever-growing set $\mathcal{F}$ will once harm the completeness property of each correct solution. Agents can deal with this problem in two ways. Firstly, an agent a_1 can delegate the filter f_j to another agent a_2 (that even does not have to be part of the path that need to be covered). This delegation consumes also a_1's r'_j resources. The utility of this delegated filter is the same as for the original filter $u'(f_j, a_2) = u(f_j, a_1)$ (if not considering price for delegation, e.g. traffic growth). Another possibility is that some of the paths remain uncovered in this case agents firstly try to cover as many paths as possible and afterwards they maximize the utility function.

Theorem 2. *The variation FAPN-E remains NP-complete.*

Proof. Following the proof of the FAPN NP-completeness it is obvious that the *distribution*, *incomplete knowledge* and *on-line task* cannot make the problem easier. The filter delegation nor the filter omission, that are used to deal with too large $\mathcal{F}$ sets, would not be used in the solution used in the original proof since they would not improve the solution of created task, e.g. in the case when $r'_j = r_j$. The filter omission will not be used since it would unnecessarily decrease the coverage.

Let us now describe our domain in the formalism presented above. The network we are protecting has a tree topology and the following items are essential for our problem:

agents : Each agent $a_i \in \mathcal{A}$ simulates a network device, such as *router, switch, hub or end user computer.* We assume that switches and routers are equipped with additional memory that can be further used.

resources :

filters : In our simulation we represent filters as *filters*. These filters can be loaded into additional memory of network devices and then filter the traffic going through the device. Each filter – filter f_j determines a set of agents it needs to *cover*. End user systems can have specified vulnerabilities and $\mathcal{A}_j$ defines a set of agents with the same vulnerability. We presume that worms can spread only between hosts with the same vulnerability.

network : All network devices are connected into the tree structure using the usual network topology.

utilities : The utility of each filter on particular device is how much it can *decrease malicious traffic* in the network, thus the sooner the traffic is filtered out the better and also the more flows go through given device the better.

5 Experiments

Several experiments were performed to show, how the Iterative ECNP (I-ECNP) can further improve the security of the system. Where I-ECNP is used more filters could be allocated into the network and more importantly their distribution in the network is adjusted for better performance therefore the system is better protected against malicious attacks. The following figures represent average from 3 test-runs using the original ECNP negotiation and 3 test-runs using the I-ECNP negotiation. All these experiments were evaluated on $\mathcal{A}$-**net** network simulation with one router and 5 subnets connected to this router. Each subnet contained 4 switchs, 15 hosts and one DHCP server.

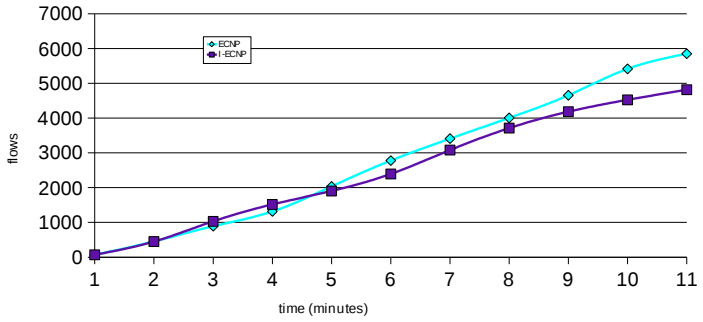

Fig. 2. Number of malicious flows reaching vulnerable hosts. This number is lower when Iterative ECNP technique is used.

Figure 2 shows comparison between number of malicious flows reaching vulnerable hosts with original negotiation used and with the new variant of Iterative

ECNP. By improving the negotiation we achieved to lower the number of flows that reach vulnerable hosts and thus can infect these hosts.

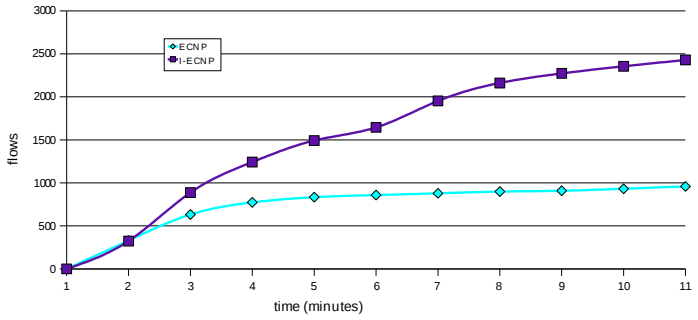

Fig. 3. Overall number of malicious flows that were filtered out by protection mechanism.

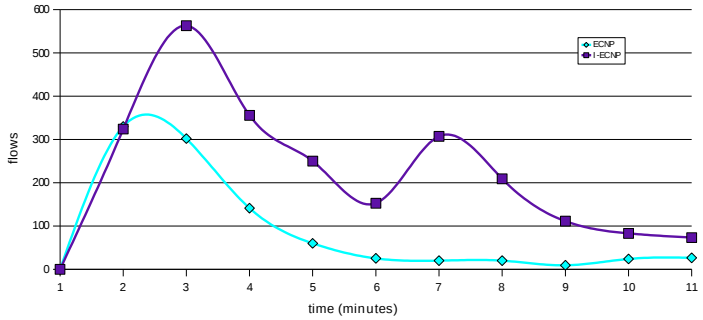

Fig. 4. Number of filtered out malicious flows per minute.

Figure 3 and 4 show how the number of successfully filtered out worm flows increased. The amount of filtered out malicious flows is higher when the new Iterative ECNP technique is used. However after certain time, the network cannot accomodate more filters and the difference between those techniques is decreasing.

6 Conclusion

Even thought network devices with the capabilities assumed in this paper are available only in laboratories, we have investigated the algorithms for filter del-

egation. Both presented algorithms are based on the negotiation using ECNP communication protocol. First one tries to delegate the filters in a greedy way, while the other runs the ECNP negotiation repetitively to choose best subset of filters to be delegated. Both algorithms were evaluated in a realistic network simulation $\mathcal{A}$-**net** and the improvement of the iterated version of ECNP was shown. We also formally described solved problem and shown that it is NP-complete.

Considering the similarity of the problem solved in this contribution and service or task allocation problem, we believe that Iterative ECNP protocol can be usefull in these domains as well.

Acknowledgement

We gratefully acknowledge the support of the presented research by Army Research Laboratory project N62558-07-C-0007.

References

1. A. Cicortas and V. Iordan. Multi-agent systems for resource allocation. In *2nd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence, SACI 2005*, 2005.
2. M. de Weerdt, Y. Zhang, and T. Klos. Distributed task allocation in social networks. In *Autonomous Agents and Multi-Agent Systems (AAMAS 2007)*. New York, NY: ACM Press, 2007.
3. R. Dijkman and et al. The state of the art in service-oriented computing and design.
4. K. Fischer, J. P. Muller, M. Pischel, and D. Schier. A model for cooperative transportation scheduling. In *Proceedings of the First International Conference on Multiagent Systems.*, pages 109–116, Menlo park, California, June 1995. AAAI Press / MIT Press.
5. M. Pěchouček, J. Tožička, Štěpán Urban, and M. Prokopová. Extending computational reflection in multiagent systems: towards autonomic computing. Technical report, The Gerstner Laboratory, Czech Technical University in Prague, 2007.
6. T. Sandholm. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter Distributed Rational Decision Making, pages 201–258. MIT Press, Cambridge, MA., 1999.
7. T. Sandholm and V. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1-2):99–137, 1997.
8. O. Shehory and S. Kraus. Coalition formation among autonomous agents: Strategies and complexity. In C. Castelfranchi and J. P. Muller, editors, *LNAI 957, From Reaction to Cognition*, pages 57–72. Springer-Verlag, Heidelberg, 1995.
9. R. G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *In IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.
10. W. E. Walsh and M. P. Wellman. A market protocol for distributed task allocation. In *In Third International Conference on Multiagent Systems*, Paris, 1998.