# A comparative assessment of peer-to-peer and server-based configuration management systems

Carlo Bellettini* and Lorenzo Capra† and Mattia Monga‡

**Abstract.** *Configuration management tools are traditionally server-based applications. To deal with the new issues emerging from the current (and future) cooperative work scenarios in which connectivity is intrinsically transient new applications based on of a fully decentralized, peer-to-peer architecture were proposed. In this paper we analyze these two architectures leveraging on Stochastic Well-Formed Nets (SWN) models, in order to compare the impact of the two alternative protocols on the collaborative work.*

## 1. Introduction

Configuration management is a very critical activity in software development. It is responsible for keeping the development of software artifacts orderly and managed. Not surprisingly, it is considered by process improvement methodologies, like CMM, as one of key practices that software development organizations should establish in their improvement strategies [11].

Traditionally, configuration management tools followed a client-server architecture. A server machine hosts a *repository* of artifacts; when one of the programmers involved in the project wants to modify an artifact, s/he must *check-out* a working copy of it on her/his machine and, when the modification is finished, the new version has to be *checked-in* again in the repository. The server is responsible for the management of the artifacts: if someone else has checked-in a new version of the same artifact, any further attempt of checking-in raises a conflict, that can be handled only by *updating* the working copy and *merging* the concurrent modifications.

This approach is suitable when a reliable and permanent network infrastructure is available to connect the participating nodes. However, it assumes the feasibility of having a central machine, set up to be accessed by all the node and reachable by a node every time it needs a check-out, a check-in, or an update operation. In order to relax this strong assumption some peer-to-peer architectures for configuration management were proposed [12, 9].

In particular, in this paper we analyze PEERVERSY[1], a peer-to-peer versioning system aimed at providing support to small groups of developers that cooperate to build software products, while forming highly dynamic virtual communities, in which people change frequently their connectivity

*Dip. Informatica e Comunicazione, Università degli Studi di Milano, Milan, Italy email: carlo.bellettini@unimi.it
†Dip. Informatica e Comunicazione, Università degli Studi di Milano, Milan, Italy email: capra@dico.unimi.it
‡Dip. Informatica e Comunicazione, Università degli Studi di Milano, Milan, Italy email: mattia.monga@unimi.it

status, not always being able to access the networking infrastructure. In PEERVERSY there are no *well known* server machines, instead the repository is distributed among the network of peers, each of which contributes to the overall logical artifact repository with the artifacts it owns. Moreover, since the network connection is intrinsically intermittent, as in the case of wireless connections, peers, in fact, are not always on-line and they may dynamically join and leave the virtual ad-hoc community, as it typically happens in mobile scenarios. They join it in impromptu meetings, where they exchange data and synchronize their work. Each peer, however, should continue to provide its functionality to the user even when it is in a disconnected stage. Thus, the supporting infrastructure should handle connections and disconnections in a seamless fashion and the configuration management application partially allows for check-out and check-in operations even if the peer is isolated from the others.

The PEERVERSY solution is based on a lazy replication of artifacts, managed by a Global Virtual Data Structure (GVDS) [5]. Consistency is maintained by a protocol similar to the one used by DNS [10], where multiple nodes record the associations between IP numbers and host names. Precisely, each peer is the *authority* for a set of items, and the copy of an artifact owned by the authority is the master copy. In addition to the master copy, peers, when performing a check-out operation to get their working copy, cache also a local copy (*replica*) of the documents for which they are not the authority to allow users to work on them even when the authority is not reachable or when the peer is disconnected from the network. In fact, a user can perform both check-in and check-out operations from the local copies of a document. The only difference between the master copy and a replica is that a check-in of a new version becomes definitive and available for all users only when the authority authorizes the changes and updates the master copy.

Initially, the authority role is assigned to the peer that enters the document into the system. The binding between authorities and peers, however, can be changed dynamically in order to improve the overall performance of the system. For instance, if node $X$ is the authority for the document $d$, but $Y$ was responsible for the last ten check-ins, $Y$ may be promoted to becoming the new authority of $d$.

Whenever a peer enters the community of peers, a reconciliation step is performed. More specifically, when $X$ gets connected, for each item $i$ for which $X$ is the authority, $X$ notifies all interested peers if a newer version of $i$ is made available (The same happens when a new checked-in copy is accepted). In such a case, the peers that own a replica of $i$ should update their copies.

Let us examine what happens when a user tries to check-out or check-in a document. Suppose that peer $X$ issues a check-out for a document $d$ whose authoritative peer is $A$ ($\neq X$). Two cases may occur:

1. $d$ is present in $X$'s local repository. In such a case the check-out operation gets a copy of $d$ from the local repository.
2. $d$ is not present in $X$'s local repository. In such a case a network search is issued to retrieve a valid copy. If no copy is found, the check-out operation fails. Otherwise, the system creates a new local replica of the document and then it checks-out a copy of the artifact from the local repository like in the previous case.

Suppose that a peer $X$ issues a check-in request for a new version of a document whose authoritative peer is $A$ ($\neq X$). We distinguish two cases:

1. $A$ is reachable by $X$. A check-in proposal is notified to $A$, which can reject the proposal or commit to making it persistent in its local part of the repository as a new master copy.

2. $A$ is not reachable by $X$. A check-in proposal is recorded in the local part of the repository hosted by $X$ unless a immediate conflict is raised. When $A$ eventually becomes available, the proposal is notified to it. $A$ can reject the proposal or commit it, by making it persistent in its local repository as a new master copy. If the item owned by $A$ is newer then the one proposed by $X$, a conflict arises and $X$'s proposal will be refused. In this case, it is $X$'s responsibility to resolve the conflict and submit a new version.

Intuitively, such a protocol provides some advantages on a server-based one. Imagine, for example, the case in which two nodes are isolated from the rest of the group: even if they cannot connect to a central server machine, they could connect one with the other. With PEERVERSY, if one of the two is the authority of document $D$, every configuration management operation is still available. If, instead, both peers are non-authoritative with respect to $D$, their replicas can be used to perform check-outs and check-ins can be cached, waiting for the first opportunity of the availability of the authority to finalize the operation. In this way people collaboration may be oblivious about actual network topology. Of course, when the number of writers is high we expect an increase in the number of conflicts arisen. In fact, in any server-based protocol, the server works as a shared point of coordination, and, *if one can assume its permanent availability,* it may be used to assure that no one works (i.e., modify artifacts) concurrently with another. However, *optimistic* protocols [8] such that the one normally used by CVS [6] do not force any locking policy, and conflicts are still possible. Therefore, in order to reduce their number, workers are advised to do, if possible (i.e., they are able to reache the server), an update before starting their work.

Since the number of nodes, their working profile, and, their connectivity profile intuitively impact on the performance of a server-based or a peer-to-peer protocol, we were interested in a comparative assessment of the two architectures. In this paper we present our findings obtained by modelling the CVS and PEERVERSY protocol with Stochastic Well-Formed Nets (SWN). The paper is organized as follows: in Section 2. we briefly describe our modelling approach, in Section 3. we present our analysis, and finally in Section 4. we draw some conclusions and propose future enhancements to our models.

## 2. The SWN model of the P2P infrastructure

In this section we present the SWN model(s) developed to analyze the impact of the peer-to-peer infrastructure realized by the `PeerVerSy` tool on (small) groups of developers freely cooperating, briefly commenting on the adopted modeling approach. Even if an exhaustive description of the models is out of the scope of the paper, let us recall here the basic concepts about the SWN formalism that are needed to motivate its use and to understand the salient points of the models presented afterwards. Refer to [2] for a complete definition of the formalism.

### 2.1. SWN basics

SWNs are a flavor of Colored Petri Nets ([7]) characterized by a structured syntax that makes it possible to detect and exploit system symmetries, thus greatly reducing the complexity of state-space based analysis techniques.

As in all High Level Petri Nets formalisms, tokens in places are associated with an identifier (color), similarly transitions are parameterized, so that different (color) instances of a given transition can be considered for enabling and firing. Arc functions associate each transition instance with a multiset of colored tokens to be withdrawn from/put into a place. Similarly a marking maps each place to a multiset of colored tokens of the corresponding domain.

Colors are built from a set of (finite) *basic color classes*. A basic color class may be (circularly) ordered, and may be *partitioned* into *static subclasses*, denoting groups of entities/components of given kind that cannot be mixed up. A *color domain*, defined as the Cartesian product of (possibly repeated) basic color classes, is associated to each place and transition. Hence the color associated with tokens in place $p$ as well as the color instances of a transition $t$, take the form of tuples of basic color class elements.

The function on an arc (denoted *I*, *O*, *H*, depending on whether the arc is *input*, *output* or *inhibitor*) connecting place $p$ and transition $t$ maps any color instance of $t$ to a multiset on the color domain of $p$. Such function takes the form of a sum of weighted tuples of *elementary functions* defined on basic color classes: the *projection*, selecting one element of a transition instance color tuple, and the *diffusion* (denoted by symbol $S$), returning the set of all elements in a given (sub)class[1]. A transition may be associated to a *guard*, that restricts the set of its admissible color instances.

A color instance $c$ of $t$ *has concession* in marking $\mathbf{M}$ iff (1) for each input place $p$ of $t$ $W^-(t,p)(c) \leq \mathbf{M}(p)$, (2) for each inhibitor place $p$ of $t$ $H(t,p)(c) >' \mathbf{M}(p)$, and (3) $guard(t)(c) =$ true (the $>, +, -$ operators are here implicitly extended to multisets).

A *priority* level is associated to each transition; priority level $0$ is reserved for *timed* transitions (represented as white boxes), while all other priority levels are for *immediate* transitions (represented as black bars), which fire in zero time.

A color instance $c$ of $t$ *is enabled* in marking $\mathbf{M}$ iff it has concession in $\mathbf{M}$ and no higher priority transition instance has concession in $\mathbf{M}$. An enabled color instance can fire, leading to the new marking $\mathbf{M}'$:

$$\forall p \in P, \mathbf{M}'(p) = \mathbf{M}(p) + W^+(t,p)(c) - W^-(t,p)(c)$$

A random *firing delay* with exponential pdf is associated to each timed transition, while weights are used to probabilistically resolve conflicts between immediate transitions with equal priority. As a result of the adopted time representation, a Continuous Time Markov Chain (CTMC) is directly derived from the reachability graph of a SWN model.

The particular syntax adopted for color domains, arc functions, and guards allows behavioral symmetries to be automatically discovered and exploited to build an aggregate state space (called *symbolic reachability graph* or SRG [3]) and corresponding stochastic process (a *lumped* CTMC). The SRG is built suitably setting a *symbolic* (i.e., parametric) initial marking, by means of a *symbolic firing rule* working directly at level of symbolic markings.

### 2.2. The SWN model of the peer-to-peer infrastructure

The models are based on the specification documents of the `PeerVerSy` infrastructure [1]. The models have been developed and analyzed by means of the `GreatSPN` graphical package [4], following

[1] the successor of a projection is also allowed on ordered basic color classes

these guidelines:

1. first the model of a traditional, client-server architecture was built
2. the complex model of the peer-to-peer architecture was then defined as a collection of submodels suitably composed: it may be considered as a *specialization* (or refinement) of (1) (according to the OO terminology), because its main component (the life-cycle of a cooperating worker) is *inherited* from in (1)
3. the other submodels of (2) represent the infrastructure that realizes the distributed repository of artifacts
4. submodels (and the overall models as well) are fully parametric, so we made explicit the relations among component parameters

The main model parameters are:

- the number of cooperating workers,
- the number of artifacts on which workers collaborate,
- the association between a set of artifacts and one worker who plays the role of authority for them.

Parameters are instantiated by properly defining the model *color domains* and *the initial marking*.

The *behavior inheritance* relation we tried to establish between the models makes us confident that a comparison between them could be reliably performed.

Concerning the abstraction level of models, we observe that very detailed models (although may be very useful for completing the existing documentation) are hardly treatable for performance analysis because of the state-space explosion they trigger.

The following issues have been considered in choosing a convenient abstraction level: (I) whether to represent the underlying communication infrastructure and resource contention, (II) whether to represent a single artifact or a set of artifacts shared by the group of developers (the former situation being better supported by the client-server architecture, while the latter one by the peer-to-peer architecture), (III) whether to consider possible failures of the communication links or of the nodes of the network (obviously the client-server architecture presents a single point of failure, while the peer-to-peer architecture is intrinsically more fault-tolerant).

In this paper we decided to not consider neither the communication infrastructure nor the possible occurrence of faults. Moreover, we restrict our analysis to collaboration based on a single artifact. These simplifications on one side allowed us to study the performance of many more configurations than those manageable on the detailed models by the `GreatSPN` tool. On the other side, coherently with the final goal of the paper (i.e., evaluating the impact of the the peer-to-peer protocol on the cooperative work), they correspond to a worst-case assumption for the peer-to-peer architecture.

The SWN model of the peer-to-peer architecture is depicted in Figure 1. It comprises three main parts: i) on the right the main peer life-cycle, ii) on the bottom the infrastructure reaction to a check-in operation, and iii) on the top the infrastructure reaction to a peer that goes online. The model of the server-based architecture (not explicitly depicted here), corresponds to the component (i) of the model in Fig.1.

Three basic color classes are used: $UID$ (whose cardinality is one of model's parameters), denoting the group of cooperating workers, $ST$ denoting the possible status (*online, offline*) of a worker, and $UPD$, denoting the status (*updated, non-updated*) of the local replica of the artifact owned by a given worker.

Timed transitions represent time-consuming activities (time being spent for decision and/or execution), while immediate transitions represent *logical* (sequences of) actions accomplished mainly system infrastructure.

All places of the net but place $Status$ have color domain $UID$: a token $\langle u_1 \rangle$ in one of these places represents that worker $u_1$ has reached a particular status of his/her life-cycle.

The set of model's timed transitions $\{checkout, startModify, endModify, update, merge, checkin\}$ have color domain $UID$: a color instance of any transition in this group represents a particular action performed by a given worker during his/her life-cycle.

In the real world, each client checks out a working copy with a specific version number, and by comparing this with the version of the last copy of the artifact on the server he is able to establish the up-to-date status. In order to exploit the symmetries of the model, we chose not to use the version numbers but to directly maintain the up-to-date status information in places $Repository$ and $WCopyUpdates$, for the repository replica and the working copy respectively.

Place $Status$ models the online status of each peer, and by changing the frequency associated to transitions $goOnlineStart$ and $goOffline$ it is possible to model the online attitude of the peers (see Section 3.1.).

Finally the place ($UpdEvent$) represents the presenceof a signal to a peer from the system that a new version of the artifact is available. We highlight here that this is an important peculiarity of the `PeerVerSy` system: it makes aware the peer as soon as possible that the working copy is not still up-to-date. Using such an information, the peers are sometime able to avoid modify actions on old versions of the artifact.

The updating of these three places is managed automatically by the infrastructure. In particular the upper part of the net represents what happens when a peer goes online. In the case that the version of such a peer is newer than those of the already on-line peers, the repository replica of such peers are automatically updated and a "New version" signal is sent to them. On the contrary in the case that the peer copy is older, is its repository that is updated and he receive the signal. This signal is thus an indication that the repository replica contains a more recent version of the checked out working copy. The bottom part of the net represents what happens when a check-in is accomplished: the working copy status of all the other peers is set to non-uptodate. The repository replica status of all the off-line peers is set to non-uptodate. A new version available signal is sent to all the online peers. All the conflicting pending check-in are then aborted ($fail$ transition). Finally is also considered the case that the check-in is accomplished by the authority when he/she is off-line. In this case no signal is

sent and all working copies and repository replica are set as non-uptodate.

## 3. Analysis of the results

### 3.1. Input parameters

Our analysis takes into account the following parameters:

1. The number of workers ($N$). We analyze the models with teams with 2 to 7 members working on a single artifact. The computational complexity of the peer-to-peer model is such that a greater number of members requires some days of CPU time on Intel Xeon 2.4GHz[2].
2. Working profile ($W$). Models assume that workers repeat cycles in which on average they work (modify the single artifact) 2 units of time and they are idle for a unit of time. When a conflict is discovered an additional unit of time is spent in the merge operation.
3. On-line profile ($O$). Every member may be on-line or off-line. This simplification is needed in order to abstract on the network topology. The server machine (in the server based model) is assumed always on-line and a worker can reach it if and only if its status is *on-line*. A peer $A$ can reach another peer $B$ if and only if the status of $A$ and $B$ is *on-line*. We characterized the on-line profile of a model with two parameters:
   (a) Connection periodicity ($OP$). How long (in time units) is a on-line/off-line cycle. This represents the dynamicity of the changes in the on-line status. We used the values of $1/2$, $1$, $2$, and $4$. $OP = 4$ means that if a worker is on-line a half of its time, on-line sessions last on average 2 time units and off-line sessions 2 time units.
   (b) On-line ratio ($OR$). How much time, on average, a worker is on-line against total time. We used ratios of $20\%$, $35\%$, $50\%$, $65\%$, $80\%$.

   Figure 2 shows the cumulative probability of finding $n$ workers on-line against different $OR$s in a setting with 5 workers.

### 3.2. Output parameters

Our assessment was based on the analysis of the following output parameters:

1. Average check-in throughput per worker ($C$). This measures how many successful check-ins were made by each worker during the simulation per time unit: the higher the better. In the peer-to-peer setting we considered separately non-authoritative peers and the authority, since one of the main advantages of the protocol is that the authority is virtually always able to perform a successful check-in operation.
2. Average merge throughput per worker ($M$). This measures how many merge every worker needed during the simulation per time unit: the lower the better, since merge operations corresponds to duplicated work. In the peer-to-peer setting we considered separately non-authoritative peers and the authority.

[2]Actually, the main problem was that data files generated by the GREATSPN tool with 8 members exceeded the maximum file size (4GB) set on our machine

3. Average number of workers that have an up-to-date version of the artifact ($U$): the higher the better.

### 3.3. Results

Figure 3 shows the average check-in (Fig. 3.a) and merge (Fig. 3.b) throughput per worker in a setting where $OR$ is $50\%$ and $OP$ is 2. The server-based case is constantly located in the middle between the authority throughput and the throughput of non-authoritative (normal) peers. In other words, the authority performs always better, since the peer-to-peer protocol favors it in check-in operations. Since is very common that one of the team members has a dominant role in the artifact management (this asymmetry is neglected in our current model), the previous result means that a peer-to-peer protocol may enhance its performances by carefully choosing the authority. Moreover, the decrease in normal peers performances is still quite acceptable (even with an increasing number of peers), given that they gain the flexibility of doing cooperative actions also when off-line.

Figure 4 shows the average check-in (Fig. 4.a) and merge (Fig. 4.b) throughput per worker in a setting where $N$ is 5 and $OP$ is 2. As expected, when the workers are mainly on-line ($OR = 80\%$) the two protocols perform very similarly. On the contrary, when the workers cannot be on-line very often ($OR <= 35\%$) the ability of working (performing check-ins) decreases significantly, except in the case of the authority, that is the only peer that gains from the peer-to-peer protocol. However, the other peers have a merge throughput largely comparable with the server-based case, thus no additional wasted work is forced by the protocol.

Figure 6 shows the average number of team members working on an up-to-date copy in the peer-to-peer case (Fig. 6.a) and the server case (Fig. 6.b) in a setting where $N$ is 5. According to this metrics, the peer-to-peer clear outperforms the server-based scenario. However, this is partially mitigated by the average check-in throughput shown in Figure 5 in the same setting.

## 4. Conclusions

A solution based on a peer-to-peer strategy offers several flexibility advantages. In particular it is possible to assign the master copy of each document to the peer who mostly uses it. In fact, when a user works on a document of which he or she owns the master copy into the local repository it is possible to work on the document without the need of continuously connecting to the network to check if a new version of the document has been released by others. A server-based approach is a perfectly acceptable solution– and in many cases a better performing solution– when the deployment of a central repository is a feasible opportunity. In fact, the peer-to-peer solution proposed by PEERVERSY was not designed to be a complete replacement for CVS or other server-based configuration management systems. Instead, a PEERVERSY-like system could be the only choice in a very dynamic environment, where nodes disconnect often and a distributed network of replicated servers cannot be afforded. Our simulations show that the performances are in general comparable with the server-based ones and in some cases even better: in fact the flexibility of the architecture may be exploited to get the best possible results from a specific connectivity scenario.

We are working on an enhanced version of our models. In fact, they currently do not take into account behavioral profiles of workers, and the possibility, in a multi-artifact context, of dealing with several authorities. We expect that this could further improve the figures of the peer-to-peer case.

## Acknowledgements

Authors would like to thank Davide Balzarotti and Carlo Ghezzi for their comments and suggestions on preliminary versions of this article.

## References

[1] Davide Balzarotti, Carlo Ghezzi, and Mattia Monga. Freeing cooperation from servers tyranny. In Enrico Gregori, Ludmila Cherkasova, Gianpaolo Cugola, Fabio Panzieri, and Gian Pietro Picco, editors, *Web Engineering and Peer-to-Peer Computing*, volume 2376 of *LNCS*, pages 235–246. Springer-Verlag, 2002.

[2] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In *Proceeding of the eleventh international conference on application and theory of Petri nets*, Paris - France, June 1990.

[3] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured petri nets. *Theoretical Computer Science*, 176(1–2):39–65, 20 April 1997.

[4] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: graphical editor and analyzer for timed and stochastic Petri nets. *Performance evaluation*, November 1995. special issue on performance modelling tools.

[5] Gianpaolo Cugola and Gian Pietro Picco. Peerware: Core middleware support for peer-to-peer and mobile systems. submitted for publication, 2001.

[6] Concurrent versions system. `http://www.cvshome.org/`.

[7] K. Jensen and G. Rozemberg, editors. *High Level Petri nets: theory and application*. Springer Verlag, 1991.

[8] H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2):213–226, 1981.

[9] Bartosz Milewsky. Distributed source control system. In *Software Configuration Management (Lecture Notes)*, pages 98–107, 1997.

[10] P. Mockapetris. Rfc 1035 (standard: Std 13) domain names–implementation and specification. Technical report, Internet Engineering Task Force, November 1987.

[11] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability maturity model, version 1.1. *IEEE Software*, 10(4):18–??, July 1993.

[12] A. van der Hoek, D. Heimbigner, and A. L. Wolf. A generic, peer-to-peer repository for distributed configuration managemenet. In *18th International Conference on Software Engineering*, page 308, Berlin - Heidelberg - New York, March 1996. Springer.

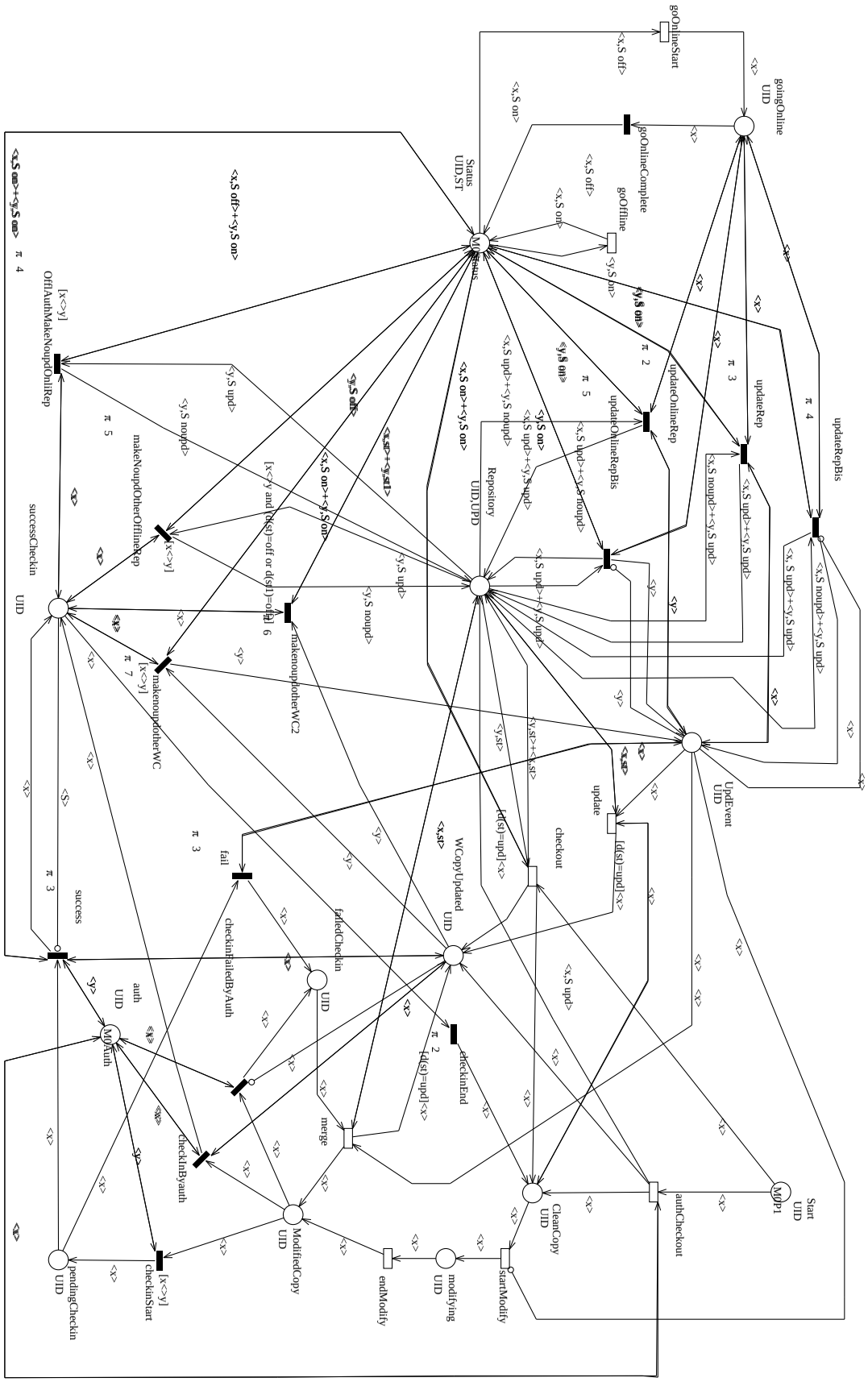


**Figure 1. The model of the peer-to-peer infrastructure**

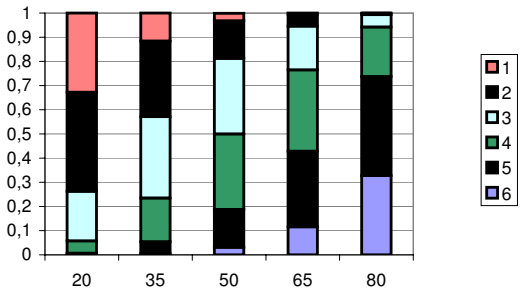


**Figure 2. Average number of on-line workers against on-line ratio**

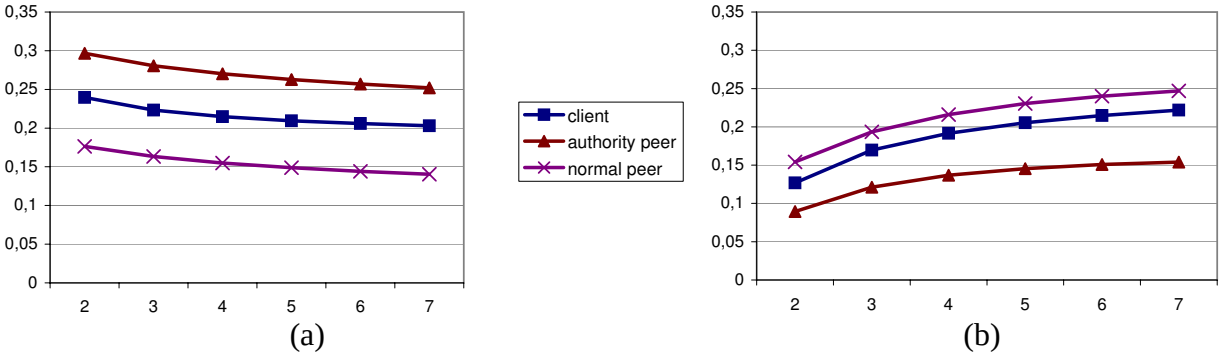


**Figure 3. Check-in and merge dynamics on increasing number of nodes**

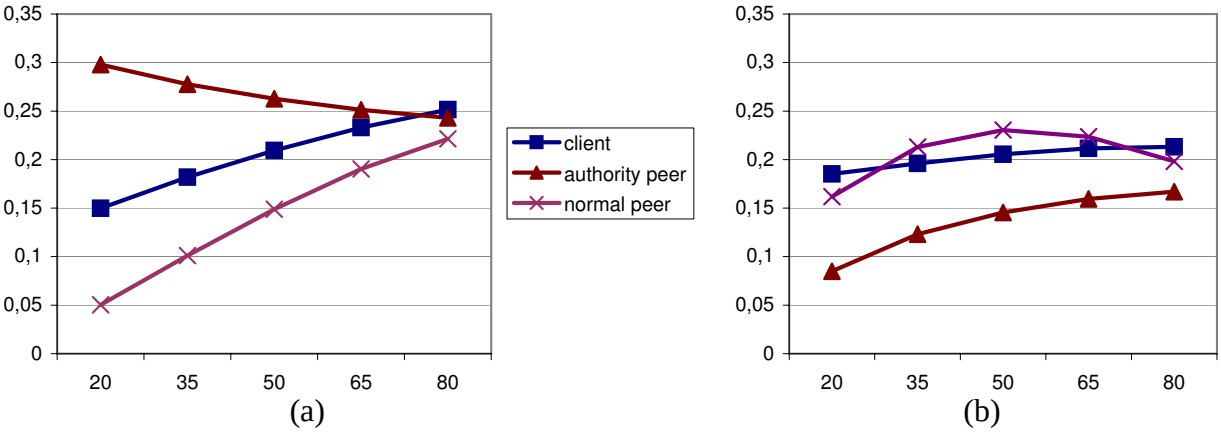


**Figure 4. Check-in and merge dynamics on increasing online attitude**

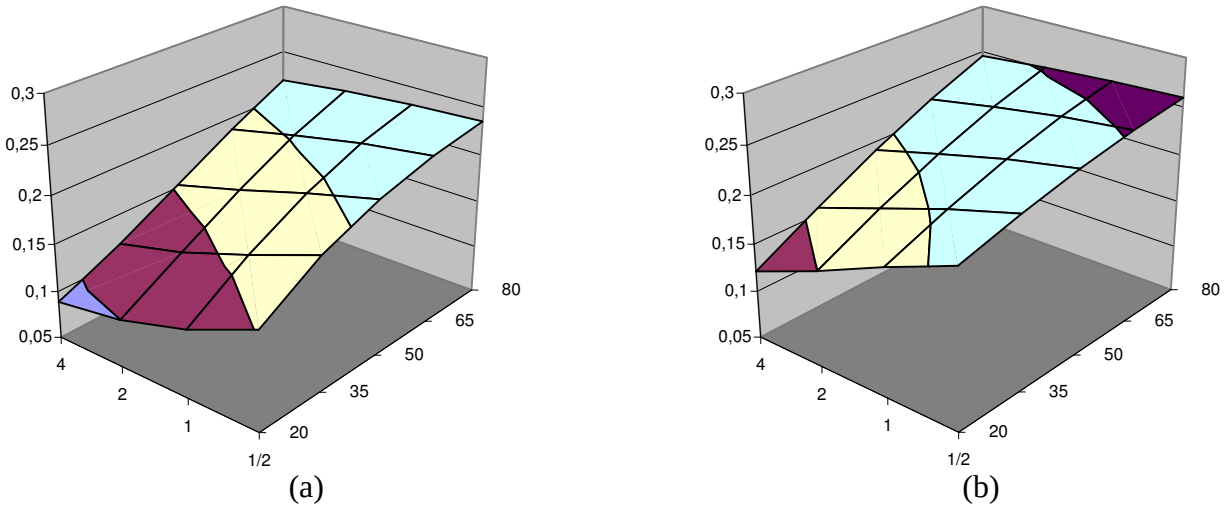


**Figure 5. Average check-in throughput against different connection profiles**

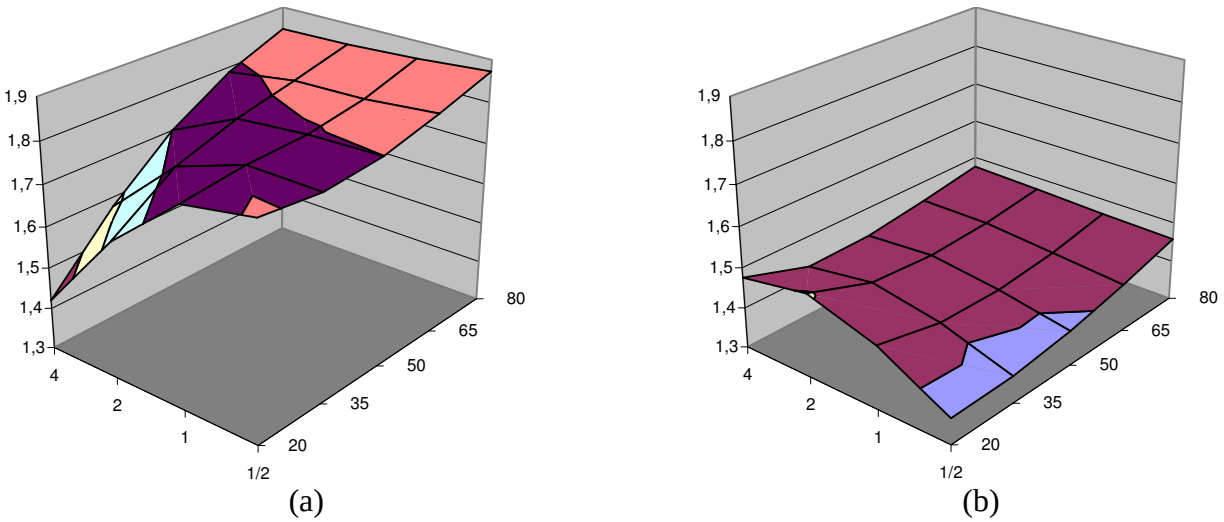


**Figure 6. Average up-to-date status against different connection profiles**