Semester project report

Utkarsh Upadhyay (utkarsh.upadhyay@epfl.ch)

June 20, 2010

Abstract

This report summarizes the work done as part of a semester project under the supervision of Prof. Rachid Guerraoui and Maxime Monod. The project addresses two distinct problems in the context of detection and blacklisting of *freeriders* in the context of high-bandwidth content dissemination with a gossip-based protocol. LiFTinG is a protocol designed to detect *freeriders*, or nodes which do not contribute their fair share of work to the system in order to save their bandwidth [5]. The first problem was to model gossip in a way which is amenable to predict how many nodes should blacklist a given node to reduce the number of packets being received by it by $x\%$. This can be useful in knowing how to send the revoke messages in LiFTinG. The second part of the project concentrated on an implementation of LiFTinG complemented with a new protocol acting as a replacement for AVMON [9]. Such a score management protocol is designed and its implementation discussed.

Page intentionally left blank

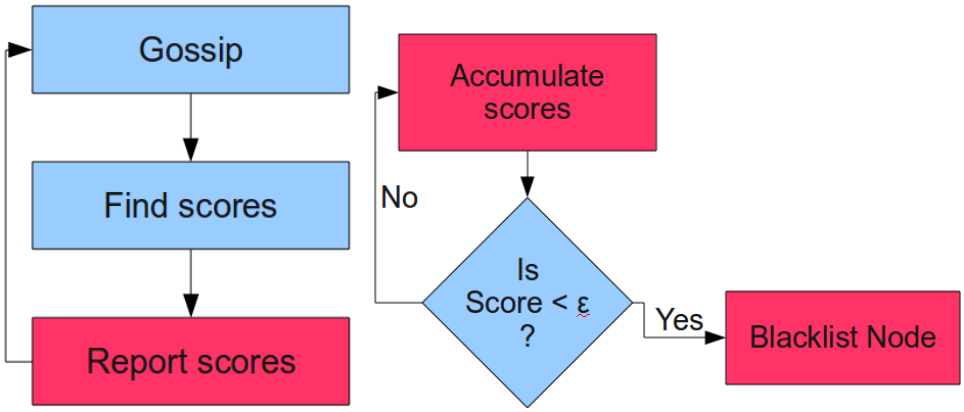
Contents

Page intentionally left blank

1 Introduction

LiFTinG is a protocol for efficiently catching and blacklisting *freeriders* in a network which requires equal participation from each node to perform at its best. The protocol is discussed in [5] and there are certain questions which remain open as to the implementation of certain components of this protocol over a network.

The LiFTinG protocol describes certain tasks a node has to perform (direct checks, cross-checking, etc.) apart from the gossip-based three-phase protocol to calculate *scores* for the nodes it is communicating with. Hence, after a round of data dissemination, some nodes in the system (called *reporters*) may generate a *score* for some other nodes (called *blamed nodes*).[1] These scores are accumulated for all blamed nodes present in the network and finally when the score of a particular node falls below a threshold, that node is *blacklisted* and is not forwarded any data anymore. The basic structure of the protocol is demonstrated in Figure 1.



(c) Routine on each *reporter* (d) Routine for each *blamed node*

Figure 1: LiFTinG protocol

This project limits itself to investigating how to concretely implement the score management, and the closely related blacklisting mechanism. Broadly, these are the boxes shown in red in the flowchart. The two parts of the projects are introduced in the Section 1.1 and Section 1.2 and the details are given in Section 2 and Section 3. The overall conclusions of the project are given in Section 4. Two appendixes are also added to explain further some aspects of the report.

1.1 Outcasts

The first part of the project deals with the problem of how many nodes should blacklist a given node such that the fraction of data reaching the node attenuates by at least x%. It is assumed that playing less than 99% of the stream is enough for the node to opt out of the network on its own. Hence, causing 1% reduction in received quality would suffice for blacklisting the node. However, with w% forward encoding, the fraction of packets begin received by the revoked node must fall by $100\% - w\%$.

The interest in this problem are manifold:

1. If the fraction is low, then only a fraction of nodes would need to blacklist the *freerider* so that reception of the stream is low enough for the node to opt out of the network on its own. It would give us a hint at how widespread should be the reach of the revoke message to ensure blacklisting[2] with a high probability.
2. If the requisite fraction of nodes maintained the score of the freerider and made individual decisions to outcast the freerider, then no revoke message would be needed.

[1] The exact mechanism of these score calculations is out of the scope of this report and is described in [5].
[2] In case of 10% FEC, it would translate to reducing reception quality of the revoked node by 10%

To this end, gossip was modeled using the classical epidemic propagation models [2] and the interaction of the outcast with the rest of the network was done using mean field equations [11].

The model predicted that about 60 to 80% of the nodes would need to revoke the outcast for the reception quality at the outcast to fall by 10% for systems with up to about 1000 nodes. These results were verified using simulations.

After verification on a smaller scale, the model was used to predict the fraction of nodes which should contact the outcast for the outcast node to receive less than 90% of the data. It was observed that the requisite fraction *decreased* as the number of nodes in the system increased. The implication of these findings are discussed later.

1.2 Score management

The design of a score management protocol for LiFTinG requires solving two separate problems:

1. **Reporting scores**: This task is to be undertaken by each node in the network as seen in Figure 1(c).
2. **Accumulating scores and revoke messages**: This task is to be undertaken for each blamed node in the network as seen in Figure 1(d).

Subsequently, this unit in LiFTinG admits various implementations:

- Centralized blaming server
- Hash based managers (as used in AVMON [9] and AVCast [10]
- Decentralized managers for different nodes

In the central blaming entity design, the second task is being done by a central server, which clearly does not scale. In the case we have hash based managers, the second task is undertaken by all the managers of any given node in the system, and as on an average, each node managed a constant ($O(1)$) number of other nodes in the system, this design scales well. However, this introduces a structure in the graph which is undesirable.

Hence, a protocol is designed which solved problem two by having each node manage $O(\log_e N)$ other nodes, but obviating the requirement of a consistent hash functions. To solve the first problem of efficiently reporting scores, a gossiping protocol is designed with some insights offered by the analysis done in the first part of the project. However, upon implementation, it was observed that this approach does not scale.

A new protocol is then designed to efficiently solve the problem of reporting the blames. The performance of the new protocol is compared with some other protocols in Section 3.4.2.

Preliminary analysis suggests that in terms of overhead, hash based manager allocation perform better than unstructured manager allocation. However, whether it is possible do better than the suggested protocol while not introducing any more structure in the network remains an open question.

2 Outcasting nodes

This section describes in detail the modeling of outcasts within the gossip protocol. As indicated above, the primary problem we look at here is how many nodes should outcast a node n such that the amount of data received by n falls by $x\%$. Knowing the answer to this question would help us in designing protocols which can provide probabilistic guarantees about what effect the dynamics of a self contained gossiping network would have on one individual node which is treated differently by different nodes in the network.

Though this individual is modeled as an outcast, it could have been a new node joining the system, or a passive monitoring entity which was probing the system to get a sample of the stream. The model developed is general enough to handle such cases.

In Section 2.1, the other models for gossiping are briefly described. The modeling developed for gossiping is detailed in Section 2.2 followed by modeling of one outcast in Section 2.3. A small extension of the model can be used to analyze the case when a constant fraction of nodes are outcast. This extension and the problems in modeling such a behavior are presented in Section 2.4. The discussion of the model predictions follows in Section 2.5 followed by the conclusions in Section 2.6.

2.1 Models for gossiping

There are multiple ways of modeling gossiping which are useful for analyzing different aspects of the dissemination in a gossip-based protocol and answering different questions. The most common ones are described here:

- **Erdös Rényi model**: This model concentrates on the path traced by a particular packet from the source to the destination and models this path as a random graph. Using the connectivity of the graph, one can prove results about the broadcast being atomic, or other properties which involve properties dependent on all nodes [8]. Modeling the behavior of one particular node among n other nodes is not easy. Hence, this model is not ideal for analyzing outcasts in an dynamic random network.
- **Macro-Discrete models:** These models keeps track of the probability distribution of the number of nodes in different states in the model at any given round [7]. Though these models can in some cases be efficiently simulated [4], this model becomes fairly intractable for a large number of nodes owing to the complex combinatorics of gossiping. The epidemic SIR model can be seen as a continuous approximation of this model which concentrates only on the mean values.

The model, initially created to study spread of epidemics, is the classical model used for studying gossiping like behaviors [2]. Apart from being tractable because of succinct expression as coupled ordinary differential equations, this model also possess the Markovian property. The behavior of one individual with respect to a large population evolving according to some Markovian rules can be modeled using mean field theory [11]. Hence, an epidemic based model was developed for gossiping. The behavior of an individual to a gossiping population was subsequently modeled using mean field equations.

2.2 The basic epidemic model

The basic model describes the *expected* delivery of one packet in a network. As all the packets are disseminated independently of each other, it is easy to see that the long term expected behavior of packets will be the same. Actually, the following two questions are equivalent:

- How many nodes should outcast a node so that it receives less than $x\%$ of data?
- What fraction of nodes should outcast a node such that the probability of 1 packet reaching the outcast is less than $x\%$?

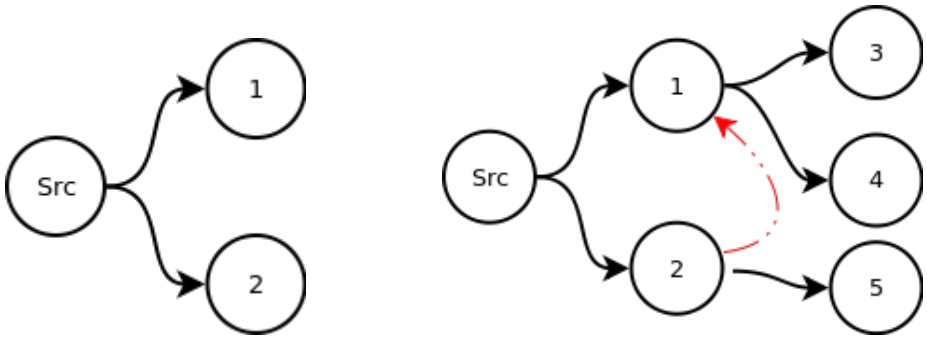
The model gives an answer to the second question by tracing the dissemination of 1 packet in the network. The set of nodes in the network are partitioned into three distinct sets which evolve over time:

- $s(t)$: The fraction of *susceptible* nodes, or the nodes which have not yet received the packet sent from the source.

- $i(t)$: The fraction of *infected* nodes, or the nodes which have received the packet, but have not forwarded the packet yet.
- $r(t)$: The fraction of *recovered* nodes, or the nodes which have received the packet, have duly forwarded it, and have finished their role in the protocol.

Capital letters would be used to denote the set of nodes which belong to the respective fractions, that is $S(t)$ would be the set of nodes which are susceptible and $s(t) = \frac{|S(t)|}{N}$, and similarly for $I(t)$ and $R(t)$.

Note here that the expressions *susceptible nodes*, *infected nodes*, and *recovered nodes* will be used to denote the respective set of nodes to remain consistent with the classical terminology. Figure 2 shows the status of the network and the susceptible, infected and recovered sets at two snapshots of the system at times t and $t + \delta$. Also, the nodes chosen by one node to communicate with in one round will be called the *fanout set* for that node in that round.



(a) $S(t) = \{3, 4, 5 \cdots\}$, $I(t) = \{1, 2\}$, $R(t) = \{Src\}$

(b) $S(t + \delta) = \{6, 7, 8, \cdots\}$, $I(t + \delta) = \{3, 4, 5\}$, $R(t + \delta) = \{Src, 1, 2\}$

Figure 2: Two rounds of gossiping with random peer selection

It can be seen that, owing to the random peer selection, the size of the set of susceptible nodes $S(t + \delta)$ does not always reduce by $f \cdot |I(t)|$ after every *complete* round, but, instead, depends also on how many nodes are present in $S(t)$.

Consider a system comprised of N nodes. Assume that the time increases with steps of $\frac{1}{N}$, so the time steps become smaller as the size of the system increases. At time t, we have $s(t)$ fraction of nodes *susceptible* to *infection*, and $i(t)$ nodes which can *infect*. At time $t + \frac{1}{N}$, exactly one node will be chosen at random. If it is in the *infected* set (with probability $i(t)$) it is removed from the *infected* set and put in the *recovered set*. The flow of packets and of nodes is shown in Figure 3(a).

As f nodes are chosen randomly out of N nodes for infection, the *expected number* of *susceptible* nodes *infected* with the message would be $\frac{f}{N} \cdot |S(t)|$. This implies that the fraction of nodes *infected* is $\frac{f \cdot s(t)}{N}$. Hence, the change in the respective fractions can be related to the state of the system at time t in the following way:

$$s(t + \frac{1}{N}) - s(t) = -\frac{i(t) \cdot f \cdot s(t)}{N} \quad (1)$$

$$i(t + \frac{1}{N}) - i(t) = i(t) \cdot \left(\frac{f \cdot s(t)}{N} - \frac{1}{N} \right) \quad (2)$$

$$r(t + \frac{1}{N}) - r(t) = \frac{i(t)}{N} \quad (3)$$

Dividing both sides by $\frac{1}{N}$, and assuming N to be sufficiently large, we get equations governing the system as:

$$\frac{ds(t)}{dt} = -f \cdot s(t) \cdot i(t) \quad (4)$$

$$\frac{di(t)}{dt} = f \cdot s(t) \cdot i(t) - i(t) \quad (5)$$

$$\frac{dr(t)}{dt} = i(t) \quad (6)$$

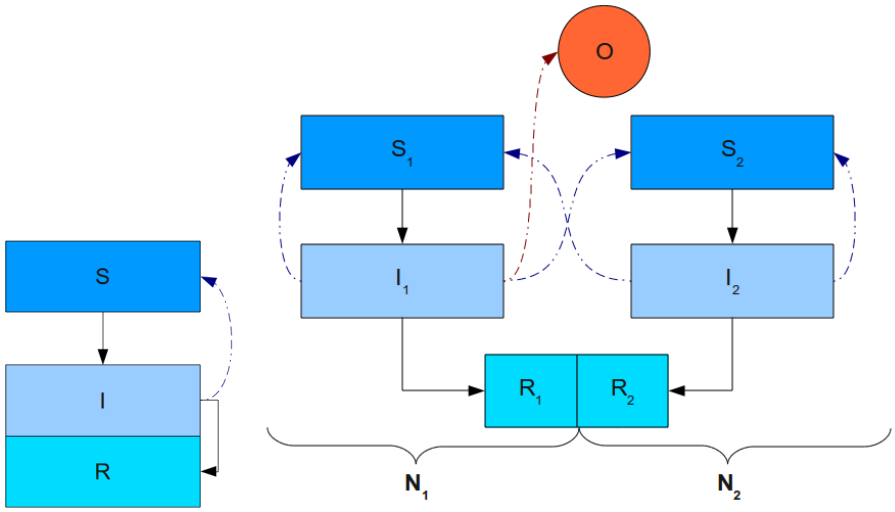
Initially, only the source has the packet (is *infected*) and all other nodes are yet to receive the message (are *susceptible*). Hence, the initial conditions for the above mentioned system of equations are:

$$s(0) = \frac{N-1}{N} \quad (7)$$

$$i(0) = \frac{1}{N} \quad (8)$$

$$r(0) = 0 \quad (9)$$

Using this model, it is possible to analyze the fraction of nodes that 1 packet is able to reach.



(a) SIR Model (b) Outcast with SIR model

Figure 3: Flow of nodes (solid lines) and flow of packets (dashed lines) in the two models

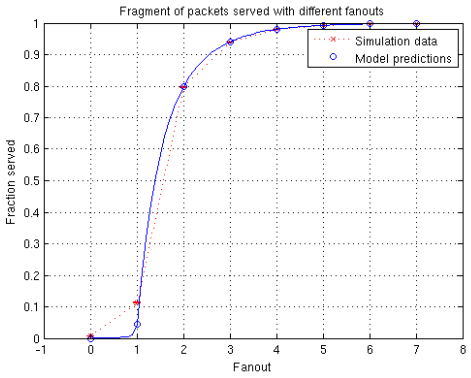
2.2.1 Verification

First, the SIR model is verified to ensure that gossiping behaves in the desired way by checking the fraction of nodes reached with different $fanouts$ and the values predicted by the model. As can be seen in Figure 4(a), except for $f = 1$ (which is a transition point for the system) the model fits the actual data fairly well. Figure 4(b) is the predicted temporal evolution of the system for dissemination of 1 packet. The behavior is, as expected, highly non-linear.

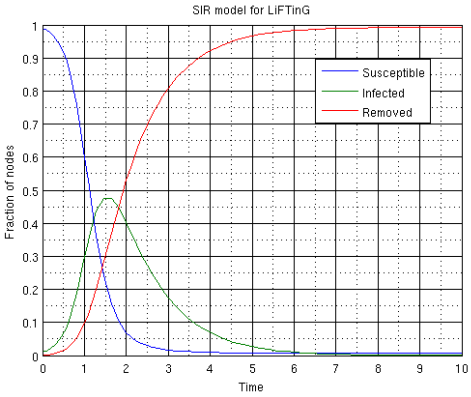
An observation which can be made from Figure 4(a) is that a fanout of 2 is sufficient to reach 80% of the nodes in the network. The implications of this property will be discussed later.

2.3 Modeling outcasts using mean field theory

In an extension of the model, one node (called the outcast node henceforth) is removed from a network of $N + 1$ nodes. The other N nodes are partitioned into two classes:



(a) Fractions reached with different f for the SIR model



(b) Temporal Evolution of system

Figure 4: Predictions of the basic SIR model

- Fraction a of nodes, which do not communicate with the outcast node; that is, they do not choose the outcast node in their *fanout set*
- The remaining nodes, which can freely communicate with the entire network

For simplifying the analysis, the following assumptions are made:

1. The *source* node is always assumed to be in the class which can communicate with the *outcast node*. This gives a slightly higher probability of the outcast being infected but the effect is negligible.
2. The *outcast* node does not take part in the protocol at all. Hence, it can only be infected, but it does not follow the protocol any further. This assumption also does not affect the behavior of the system much, if the number of nodes is large.

Gossiping over the network can be seen as a Markovian decision evolutionary game as all the decisions made (the choice of f nodes in the fanout set) and the next state of the network ($S(t + dt)$, $I(t + dt), \cdots$) are independent of the past states of the network. Hence, the outcast can be modeled as an individual using mean field equations as described in [11].

Define N_1 to be the set of nodes which can communicate with all nodes, and let N_2 be the set of nodes which have revoked the outcast. It is clear that $|N_1| + |N_2| = N$. Now define 5 different sets (instead of just 3 set of nodes for the SIR model) of nodes. The capital letters denote the number of nodes, while the small letters would mean the corresponding fraction of nodes. The description of each class is given in Table 1.

Set	Description
$I_1(t)$	Number of nodes which are *infected* in N_1
$S_1(t)$	Number of nodes which are *susceptible* in N_1
$I_2(t)$	Number of nodes which are *infected* in N_2
$S_2(t)$	Number of nodes which are *susceptible* in N_2
$R(t)$	Number of N nodes which have been *removed*

Table 1: Description of variables in the model

The evolution of the system is the same as in the SIR model, with some *cross* terms introduced owing to the infections in the set $S_1(t)$ done by nodes in set $I_1(t)$ and vice versa, as shown in Figure 3(b). The differences in the superficial complexity in the models can be visualized by comparing Figure 3(a) and Figure 3(b).

In addition to these variables, define another attribute of the system, which is a boolean variable $\Lambda(t)$ defined as:

$$\Lambda(t) = \begin{cases} 0 & \text{if the outcast has not been infected} \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

These sets with the boolean variable completely define the state of our system.

At each time step from t to $t + \frac{1}{N}$, one node x is randomly chosen from the N nodes. There are three cases to be analyzed:

1. $x \in I_1(t)$: In this case, it will infect f nodes (chosen from $N+1$ nodes) and will be removed from $I_1(t)$ and added to $R(t + \frac{1}{N})$. A node can be infected only if it either belongs to $S_1(t)$ (in which case it is shifted to $I_1(t + \frac{1}{N})$) or to $S_2(t)$ (in which case, it is shifted to $I_2(t + \frac{1}{N})$)
2. $x \in I_2(t)$: Similar to case 1, but the choice of f nodes is only limited to N nodes in the network, without the outcast.
3. If x does not belong to either, then no action is taken.

Now the expected changes made to these variables can be written as:

$$E[I_1(1 + \frac{1}{N}) - I_1(t)] = \frac{I_1(t)}{N} \cdot \left(f \cdot \frac{S_1(t)}{N+1} - 1 \right) + \frac{I_2(t)}{N} \cdot f \cdot \frac{S_1(t)}{N+1} \tag{11}$$

$$E[S_1(1 + \frac{1}{N}) - S_1(t)] = -\frac{I_1(t)}{N} \cdot f \cdot \frac{S_1(t)}{N+1} - \frac{I_2(t)}{N} \cdot f \cdot \frac{S_1(t)}{N} \tag{12}$$

$$E[I_2(1 + \frac{1}{N}) - I_2(t)] = \frac{I_1(t)}{N} \cdot f \cdot \frac{S_1(t)}{N+1} + \frac{I_2(t)}{N} \cdot \left(f \cdot \frac{S_1(t)}{N+1} - 1 \right) \tag{13}$$

$$E[S_2(1 + \frac{1}{N}) - S_2(t)] = -\frac{I_1(t)}{N} \cdot f \cdot \frac{S_2(t)}{N+1} - \frac{I_2(t)}{N} \cdot f \cdot \frac{S_2(t)}{N} \tag{14}$$

$$E[R(1 + \frac{1}{N}) - R(t)] = \frac{I_1(t) + I_2(t)}{N} \tag{15}$$

Also, to see the changes in the boolean variable $\Lambda(t)$, we know that $\Lambda(t) = 0$ if and only if the outcast has not been infected in the system at any previous time step. Also, we know that the outcast can be chosen only by nodes present in set $I_1(t)$. Hence, we have:

$$\begin{aligned} E[\Lambda(1 + \frac{1}{N}) - \Lambda(t)] &= Pr[\Lambda(t) = 0] \cdot (\text{Prob. of infection}) \\ &\quad + Pr[\Lambda(t) = 1] \cdot 0 \end{aligned} \tag{16}$$

$$\text{Prob. of infection} = i_1(t) \cdot \left(1 - \left(\frac{N}{N+1} \right)^f \right) \tag{17}$$

$$\approx i_1(t) \cdot \left(\frac{f}{N+1} \right) \tag{18}$$

$$\text{Let } Pr[\Lambda(t) = 1] := p \tag{19}$$

$$= E[\Lambda(t)] \tag{20}$$

$$\text{Then, } Pr[\Lambda(t) = 0] = 1 - p \tag{21}$$

$$\text{Equation 16 gives: } \frac{E(\Lambda(t + \frac{1}{N})) - E[\Lambda(t)]}{(1 - E[\Lambda(t)])} = i_1(t) \cdot \frac{f}{N+1} \tag{22}$$

$$\frac{1}{(1-p)} \cdot \frac{dp}{dt} = i_1(t) \cdot \frac{f}{N+1} \tag{23}$$

Now, we can describe the state of the system as:

$$M^N(t) := [i_1(t), s_1(t), i_2(t), s_2(t), r(t), \Lambda(t)] \tag{24}$$

$$= [\frac{I_1(t)}{N}, \frac{S_1(t)}{N}, \frac{I_2(t)}{N}, \frac{S_2(t)}{N}, \frac{R(t)}{N}, \Lambda(t)] \tag{25}$$

Then the drift equation, which denotes the *expected* change in the state of the system in one time step, can be written as:

$$\vec{f}^N(t) := E\left(M^N(t + \frac{1}{N}) - M^N(t) \mid M^N(t) = \vec{m} \right)$$

If the probability distributions of the states of the system follow certain technical constraints (given in [11]), then the expected change in microscopic quantities $(I_1(t), S_2(t), \cdots)$ can be related to the rate of change of the macroscopic quantities $(i_1(t), s_2(t), \cdots)$ by dividing by the time step dt, which is $\frac{1}{N}$ in our case. These constraints reduce to saying that the final state of the system converges as time tends to ∞ and that the changes in the system at each step are not *very large*. Both these constraints are met by our model system. Hence, the differential equations for the states (which are $i_1(t), i_2(t), \cdots$) can be obtained by:

$$\begin{aligned} \text{Let } M(t) &:= \lim_{N\to\infty} M^N(t) && (26)\\ f(M(t)) &= \lim_{N\to\infty} \frac{f^N(t)}{\frac{1}{N}} && (27)\\ &= \lim_{N\to\infty} N \cdot f^N(t) && (28) \end{aligned}$$

Using these techniques, the final equations obtained are the following:

$$\begin{aligned} \frac{di_1}{dt} &= i_1(t)\cdot(f\cdot s_1(t)-1) + i_2(t)\cdot f\cdot s_1(t) && (29)\\ \frac{ds_1}{dt} &= -f\cdot s_1(t)\cdot(i_1(t)+i_2(t)) && (30)\\ \frac{di_2}{dt} &= i_1(t)\cdot f\cdot s_1(t) + i_2(t)\cdot(f\cdot s_1(t)-1) && (31)\\ \frac{ds_2}{dt} &= -f\cdot s_2(t)\cdot(i_1(t)+i_2(t)) && (32)\\ \frac{dr}{dt} &= i_1(t)+i_2(t) && (33)\\ \frac{dp}{dt} &= (1-p)\cdot i_1(t)\cdot f && (34) \end{aligned}$$

As was assumed above, the initial conditions for the system are such that only one node (source) is infected and the rest of the nodes are all susceptible. Also, fraction a of nodes belong to the set N_1 and the rest to set N_2.

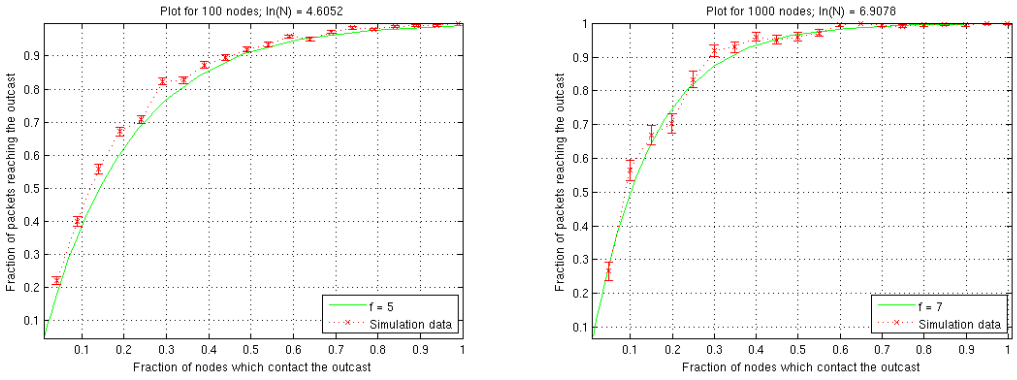
$$\begin{aligned} i_1(0) &= \frac{1}{N} && (35)\\ s_1(0) &= a - \frac{1}{N} && (36)\\ i_2(0) &= 0 && (37)\\ s_2(0) &= 1-a && (38)\\ r(0) &= 0 && (39)\\ \Lambda(0) &= 0 && (40)\\ \implies p(0) &= 0 && (41) \end{aligned}$$

The quantity of interest would be $\Lambda(\infty)$ or $p(\infty)$, which is the probability with which the outcast gets infected.

2.3.1 Verification

To test the model with the outcast, multiple experiments were run while varying number of nodes in the system. Two of these cases are presented here, for $N = 100$ nodes in Figure 5(a) and for $N = 1000$ nodes in Figure 5(b). These show a close fit of the simulation data by the model predictions.

Both these experiments were run without forward error correction (FEC). Enabling 10% FEC was expected to result in more than 99% reception when the data received by the outcast node was over 90%. The experiments were then run with FEC turned on and the results obtained are shown in Figure 6. It can be noticed that there is a *slight* jump near the point where the model predicts 90% reception (more visible in Figure 6(a)). If the node receives more than 90% of the data, it should be able to reconstruct almost all the packets, or the simulation results should indicate a flat

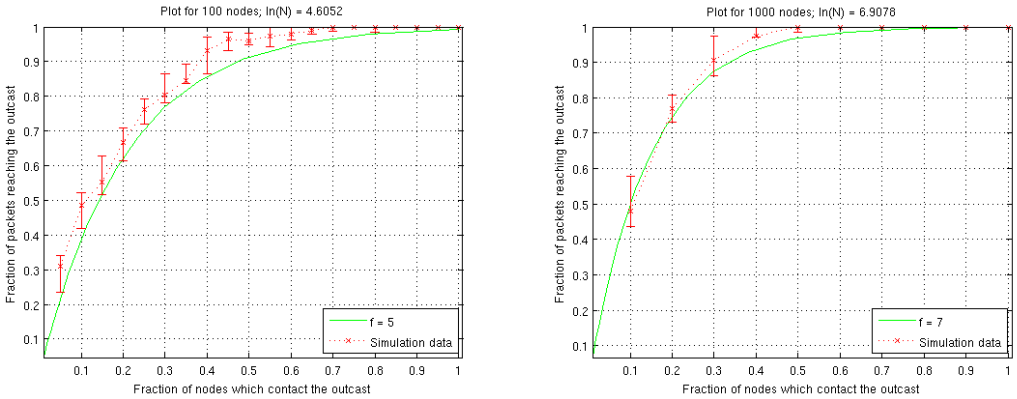


(a) 100 nodes, 1 outcast

(b) 1000 nodes, 1 outcast

Figure 5: Verification of model predictions (without any forward error correction)

curve after the 90% reception point (more visible in Figure 6(b)). The increase is much less when the fraction of nodes contacting the outcast is less than the requisite fraction to provide with 90% of the data, and can be attributed to variance in packet delivery.



(a) 100 nodes

(b) 1000 nodes

Figure 6: Comparison of model predictions with a stream with forward encoding, the nodes receive data which is within 10% of the data predicted by the model

2.4 More than one outcast

The above analysis deals with the outcast as an individual entity whose involvement alone does not perturb the system by a large amount. However, if a large fraction of nodes are outcast from a network, they may have a significant impact on the system. Under certain assumptions, these outcast nodes can be modeled using the basic SIR model itself.

If the number of nodes is $O(1)$, then their impact on the network clearly will not be significant as the number of nodes tend to infinity and they can be modeled independently as individual as suggested in the previous section.

However, if the number of outcast nodes scales as $O(n)$, then they can be modeled within the SIR model itself. Consider the case when a fixed fraction b of nodes are outcast. For simplicity, it is assumed that they behave in the same manner as honest nodes. It is easy to modify the model to incorporate any changes in in behavior they might collectively have (lower fanout, biased partner selection, etc.)

In this case, we will have three sets of nodes. N_1 will be the nodes which communicate with everyone. N_3 will be the set of outcast nodes, and N_2 will be the set of nodes which do not communicate with any outcast nodes. Note that this requirement also can be relaxed to saying that these nodes communicate the outcasts with a lower probability. This is not unlike modeling the behavior of heterogeneous gossiping-based protocol [3].

In this case we will have 7 classes described in Table 2. The flow of packets and nodes is shown in Figure 7.

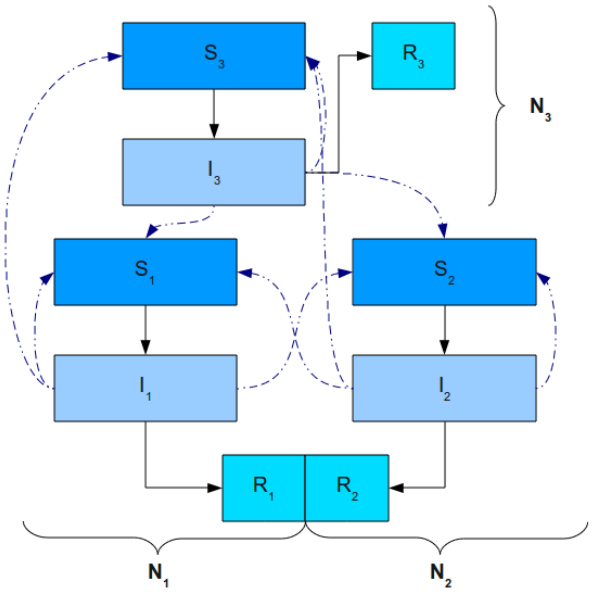


Figure 7: SIR model with three classes, R_1, R_2, R_3 are shown separately only for clarity

Set	Description
$I_1(t)$	Number of nodes which are *infected* in N_1
$S_1(t)$	Number of nodes which are *susceptible* in N_1
$I_2(t)$	Number of nodes which are *infected* in N_2
$S_2(t)$	Number of nodes which are *susceptible* in N_2
$I_3(t)$	Number of nodes which are *infected* in N_3
$S_3(t)$	Number of nodes which are *susceptible* in N_3
$R(t)$	Number of N nodes which have been *removed*

Table 2: Description of variables in the case there are $O(n)$ outcasts

As was the case for the original SIR model, the equations can be worked out in almost the same manner:

$$\frac{di_1}{dt} = i_1(f\,s_1 - 1) + i_2 f \frac{s_1}{1-b} + i_3 f\,s_1 \quad (42)$$

$$\frac{ds_1}{dt} = -f\,s_1(i_1 + \frac{i_2}{1-b} + i_3) \quad (43)$$

$$\frac{di_2}{dt} = i_1 f\,s_2 + i_2(f\frac{s_2}{1-b} - 1) + i_3 f\,s_2 \quad (44)$$

$$\frac{ds_2}{dt} = -f\,s_2(i_1 + \frac{i_2}{1-b} + i_3) \quad (45)$$

$$\frac{di_3}{dt} = i_1 f\,s_3 + i_3(f\,s_3 - 1) \quad (46)$$

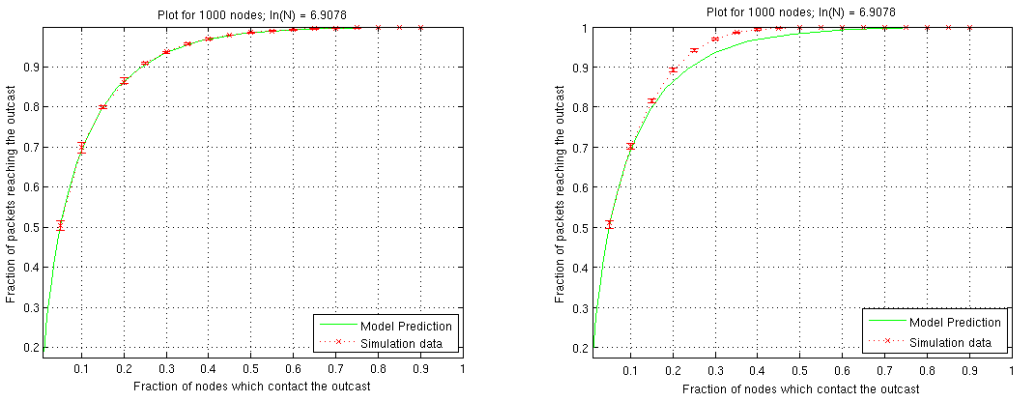
$$\frac{ds_3}{dt} = -f\,s_3(i_1 + i_3) \quad (47)$$

$$\frac{dr}{dt} = i_1 + i_2 + i_3 \quad (48)$$

However, it is not clear how one may model a system where the number of outcasts grows but not as $O(N)$.

2.4.1 Verification

As the model for a fraction of outcast nodes also closely fits the SIR scheme itself, it was also tested with the same methodology and the results are shown in Figure 8.



(a) Fraction of packets reaching the outcast fraction (10%) of nodes without FEC

(b) Fraction of packets reaching the outcast fraction (10%) of nodes **with 10% FEC**

Figure 8: Fraction of packets reaching the outcast nodes when 10% nodes are outcasts

The results are given for both with and without FEC. It can be noticed again that the effect of FEC is negligible if the fraction of packets arriving at the outcast nodes is less than 90%, and becomes significant after the reception grows above 90%.

2.5 Model predictions for the outcast node

Since the model predictions and simulation results are fairly close, the model can be used to predict the behavior of the system for larger number of nodes which cannot be simulated easily. As per the modeling assumptions, we expect the model to mimic reality more closely as the number of nodes in the system tend to infinity.

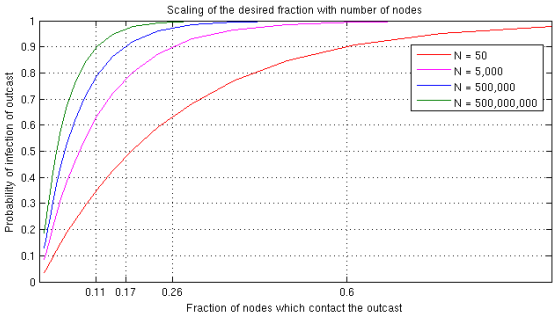
One particular feature which can be noticed in the graphs shown in Figure 5 is that the fraction required to obtain 90% of the data varies with different values of N. From the point of view of the model, the changing parameters are:

1. Initial conditions ($s(0) = \frac{1}{N}$, etc.)

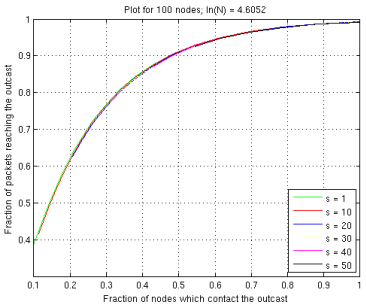
2. Fanout: $f \approx \log_e N$

The to see the effect of the initial conditions (number of sources) on the model, the different states were predicted keeping the $fanout$ parameter the same. The model predictions are shown in Figure 9(b).

Then the number of sources was kept the same and the evolution of the system plotted for increasing N, which resulted in the results shown in Figure 9(a). These cannot be empirically verified owing to limits on capabilities of the simulator.



(a) Scaling of the requisite fraction with N



(b) Scaling of the requisite fraction with different number of sources

Figure 9: Fraction of nodes which need to contact the outcast to deliver 90% packets to it

2.6 Conclusions

The conclusions which can be drawn from the analysis are the following:

1. To ensure that a node receives than then 90% of the data, the fraction of nodes which need to remove it increases with increasing N. Actually, the fraction increased with the size of the fanout set, as shown in Figure 9(a) and in Table 3. This also indicates that the fraction of

Number of nodes in network	Fanout size	Required Fraction
50	4	40%
5,000	9	74%
500,000	14	82%
500,000,000	21	89%

Table 3: Fraction of nodes which needs to stop communicating with the outcast to get 90% of the data, as shown in Figure 9(a)

   nodes which need to communicate with a newcomer for the newcomer to receive 90% of the data grows less as the size of the fanout set increases.

2. The other observation which can be made from Figure 4(a) is that a large fraction of nodes can be reached on average using a relatively small fanout, irrespective of the number of nodes in the network. A fanout of 2 would reach about 80% of the nodes and a fanout of 3 would reach more than 90% of the nodes in the network. Hence, a revoke message sent with a fanout size of 3 would be enough in a network of 500,000,000 nodes to ensure that more than 89% of the nodes revoke the blacklisted node on average.
3. The model has been verified and can be used for other investigations.

Hence, the model described answers conclusively the original questions and opens doors for further interesting investigations. Some possibilities are suggested in Section 4.

3 Implementing score management for LiFTinG

The second part of the project dealt with the problem of designing a protocol to solve the score management problem which can be implemented efficiently and could be scaled to a large number of nodes without overburdening any particular node. The problem of maintaining a score is a two pronged problem:

- Each node should be able to report the score it calculates for other nodes in the system.
- For each blamed node, its scores should be accumulated and the node should be blacklisted if its score falls below a certain threshold.

This section details some of the implementations which can solve this problem of score management, starting with the simplest solution to implement in Section 3.1 to the current state of art protocols in Section 3.2. Section 3.3 takes a step back and analyzes the desirable properties for such a protocol and describes a simple solution and its problems. In Section 3.4, a compromise between the different protocols is described which takes us one step closer to an ideal solution to the problem. Finally, some conclusions are discussed in Section 3.5.

3.1 Central blaming entity

This is the simplest score management to implement. One particular node manages all other nodes in the system as shown in Figure 10. All the nodes know about C and report the calculated scores for other nodes to this node. The central server **C**, in turn, keeps track of all the scores for all the nodes and sends out the revoke message as soon as the scores fall below a certain threshold for each node it manages.

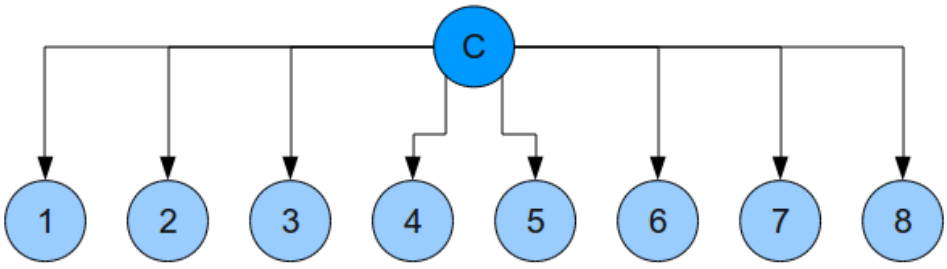


Figure 10: Central server managing all nodes in the system

The drawback of the system is evident from the figure itself, that it is clearly not scalable. As the number of nodes increase, the memory overhead on the server will increase linearly as $O(N)$. Also, after a round of gossip, each node will send the score values for $f \approx O(\log_e N)$ back to this server. Hence, the bandwidth used per gossip round at the server increases as $O(N \log_e N)$. This is clearly unacceptable.

3.2 AVMON

An alternative is to let the nodes be manager by a fixed set (of size m) of other nodes. This can be done if the manager/managee relationship is a function of the hash values of the IDs of the nodes themselves. This approach is shown in Figure 11 where the relationship is $hash(manager) < hash(managee) \leq hash(manager) + 2$, hence $m = 2$.[3]

This approach had clearly certain advantages to offer over having a central server, the first being scalability itself. Each node has to manage approximately m nodes to ensure that each node has exactly m managers. Note that this is guaranteed only with a high probability in a population of finite number of nodes and non-ideal hash function. This is the approach used for availability monitoring in AVMON [9]. The protocol to solve the two problems is the following:

1. **Reporting the scores:** Each *reporter* requests the *blamed node* for a list of its managers and upon receiving the list, sends a unicast message to each manager.

[3]In practice, owing to non uniformity of hash functions and finite number of nodes, the number of managers is usually not the same for all nodes in the system.

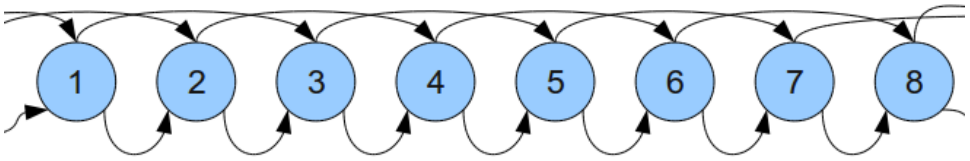


Figure 11: Hash based manager selection, each node gets two managers and has two managers

2. **Accumulating scores and blacklisting:** Each *manager* accumulates score for its *managee* nodes and initiates a revoke message to be multicast as soon as the score value falls below a certain threshold .

However, as is evident in Figure 11, having such a manager/managee relationship introduces a structure in the graph, which is not desirable for a gossip-based protocols.

3.3 Properties of an ideal score management protocol

In an ideal case, we would want to have a score management system which scales well with the size of the system and which does not introduce a structure in the graph. Additionally, simplicity in implementing the protocol is another desirable property of such a system. Nevertheless, scalability being the primary requirement, any central server solutions can be ruled out at this stage though they are the simplest to implement.

A possible way of implementing the score management system would be to have independent nodes in the system whose only task is to maintain the scores. However, this approach would require the monitoring nodes to be aware of the joining and leaving nodes and the number of these nodes will need to increase as $O(n)$ too. Hence, it is desirable that we can let the nodes themselves be the score managers of other nodes in the network. This introduces functional uniformity among the nodes and would make deployment of LiFTinG protocol considerably easier.

For such a system, for each node x, we will have two sets, namely:

- **PS(x)**: The *Ping set* of node x, or the managers of x, and,
- **TS(x)**: The *Target set* of node x, or the nodes x is a manager of, or the managees of x.

However, allowing nodes to manage scores for each other introduces the problem of having *colluding nodes* in the system. Such a protocol must have the property that if there are a fraction of colluding nodes in the system, then they do not do allow score forging (at least with high probability, the protocol should prevent this from happening).

The authors in [9] have investigated what properties such a monitoring system must have. These are:

1. **Consistency**: $y \in PS(x)$ must not change with changes in the system size (with churn).
2. **Verifiability**: Given three nodes (x, y, z), z should be able to verify whether $y \in PS(x)$.
3. **Randomness**: For all nodes x, $PS(x)$ must contain independent and identically (uniformly) distributed nodes.
4. **Discoverability**: Node x should be able to discover $TS(x)$ and $PS(x)$ quickly.
5. **Load balancing & Scalability**: The overhead for discovering $PS(x)$ and $TS(x)$ should be distributed evenly on the entire network and these overheads must be scalable.

AVMON has all these properties and runs as a separate protocol independent of gossiping. However, there are certain properties which are not required for score management in LiFTinG. The score management system for LiFTinG should ideally utilize the following properties:

- Knowledge of $PS(x)$ is not a necessity for a node x (This may be useful, or harmful)
- Nodes **only** provide negative feedback for other nodes. Also, there will never be any false negative feedback sent in a rational system [6].

Keeping these in mind, a protocol was designed which did not depended on any relationship between the node IDs.

3.3.1 Completely unstructured managers

The suggested solution is allowing nodes to choose their managers uniformly at random on the whole set, as illustrated in Figure 12.

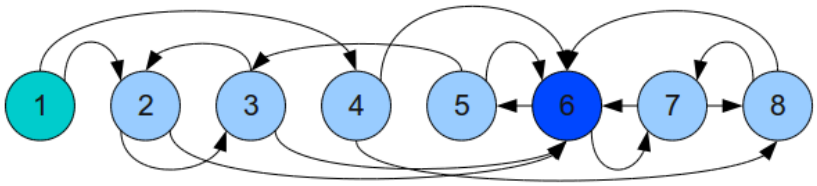


Figure 12: Each node choses two managers uniformly at random, Node 1 does not have any manager, Node 6 has 6 managers

This introduces another issue of ensuring with a high probability that all the nodes are covered by (say) m managers. This can be ensured by increasing the number of managers each node chooses. Preliminary analysis suggests the conjecture that each node will only need to manage about $O(\log_e N + f(m))$ nodes to ensure that all nodes have at least m managers. For the detailed empirical analysis, see the attached Appendix.

If we assume that the conjecture stated above holds, then a fairly simple solution to the problem of score management emerges. The managers are alloted by each node randomly selecting $(\log_e N + m + O(1))$ nodes it wished to manage, thus ensuring with high probability that each node has m managers. This information is kept private by each node, so nodes do not know the managers of each other.

When *reporters* have to send scores for *blamed nodes*, they gossip the blame values throughout the network. When a node receives such a blame message for a node which it is managing, it accumulates that score. It was observed in Section 2.2.1 that with a fanout of about 3, close to 95% of the nodes can be reached on average. Hence, it should be possible with a small fanout to reach most of the managers. This solution was implemented to check for the bandwidth overhead incurred by the nodes.

It was observed that with increasing N, the overhead of these gossiped messages increased. In part, this behavior can be explained by seeing that overhead of n nodes gossiping blame messages once per p rounds is equivalent to one single source which gossips $\frac{n}{p}$ messages per gossip round. Hence, if the blame gossiping period p is $O(1)$ (or that it remains constant), then the overhead of blame messages would scale almost as $O(N)$. Also, increasing the gossip period would result in:

- Fewer score reports as the system size increases. This would have an adverse affect on the speed with which freeriders are caught in a network with a large number of nodes.
- Still larger message sizes because the blame values would accumulate for more nodes in the blame gossiping period.

These artifacts are clearly undesirable and prevent the protocol from scaling to a large number of nodes.

3.4 Alternate solution: Caching addresses

The problem of overhead in the previous solution was caused because of gossiping the blame messages throughout the network. This problem can be solved if we limit the *hop* counts of the gossiped message while ensuring reliable delivery of the message. One possible solution with this observation in mind is presented here.

For this protocol, each node first chooses k other nodes, whose managers it will *remember*. Then nodes select the managee nodes, as in the previous case, and *broadcast* this information to other nodes (via gossiping). Assume that on average, each node has m managers. This way, each node populates its table of managers for the k nodes it had initially selected as shown in Table 4.

Note that this is during the initialization of the system itself, and, hence, this overhead is only incurred once. How to deal with nodes dynamically joining and leaving the system would be discusses later.

(a) Address table for Node 3

Managers of:	Manager list
1	-
4	{1}
8	{4, 7}

(b) Address table for Node 4

Managers of:	Manager list
2	{1,3}
8	{4, 7}
6	{2, 3, 4, 5, 7, 8}

Table 4: Sample address tables after population for the network shown in Figure 12, Nodes choose $k \approx \sqrt{(8)} \approx 3$ random nodes whose managers they remember

Whenever a reporter needs to communicate scores for a node x, it first checks if the address of the managers of x is present in the k entries in its own address table. If yes, then the it sends a unicast message to each of the managers of x. If not, the score is sent to l other nodes in the system. Upon reception of such a relayed message, a node looks through its own list of k addresses and sends a unicast message to the managers of x if the entry is found. *Otherwise, the relayed message is discarded.* This prevents wasteful gossiping of messages while still remaining resistant to colluding nodes.

Let π denote the probability of a blame message reaching the managers of x. A message will not reach the managers of x if none of the l nodes have the address of managers of x in their address book.

$$\pi = 1 - \left(1 - \frac{k}{N}\right)^l \tag{49}$$

$$\approx 1 - e^{-\frac{l \cdot k}{n}} \tag{50}$$

If we choose $l \cdot k = c \cdot n$, π can be made close to the order of probability of successful transmission of a packet on the network.

In this case, we have:

$$\text{B/W overhead} : = \overbrace{\frac{k}{N} \cdot \left(f + \left(1 - \frac{k}{N}\right) \cdot f \cdot l\right)}^{\text{Own + Relayed blames to unicast}} + \underbrace{\left(1 - \frac{k}{N}\right) \cdot l}_{\text{Own blames to relay}} \tag{51}$$

$$= f \cdot \left(l + \frac{k}{N} - \frac{l \cdot k}{N} \cdot k\right) \tag{52}$$

Also, in this case, the memory overhead per node is $O(m \cdot k)$, as address of m managers of k nodes is kept in memory by each node.

If we choose $l \approx c \cdot \sqrt{N}$ and $k \approx \sqrt{N}$, then the bandwidth overhead is $O(f \cdot \sqrt{N}) = O(\sqrt{N} \log_e N)$ and the memory overhead is $O(c \cdot \sqrt{N} \cdot \log_e N)$.

3.4.1 Nodes joining and leaving the network

After the manager/managee relationship has been set up, nodes will need to regularly check their managee nodes regularly to purge the system of dying nodes. Also, newcomers will need to populate their addresses books of $O(\sqrt{N})$ other nodes and need to get m managers for themselves.

The JOIN algorithm, which takes care of these requirements, involves the following steps:

1. The joining node x will inform the Bootstrap node in the system about its joining.
2. The Bootstrap node selects m managers for node x and informs them of x's presence. The m managers add entry x in their score tables and multicast the message $IMAN(\text{x})$[4].
3. The Bootstrap node asks t nodes in the system to share their address book with node x. There is a possibility of an attack at this point, which is that a colluding node will try to populate the manager address book by sending a false list of managers of their own and colluding nodes.

[4] $IMAN$ is an abbreviation of *I MANage*

Some analysis would be needed here to determine how collusion resistant can the system be made without having a large t, but it is expected that the probability of such a contamination would remain low for even a reasonable number of colluding nodes.

4. The node x shuffles the $t \cdot c \cdot \sqrt{N}$ entries it receives and fills its address books with $c \cdot \sqrt{N}$ random entries (along their managers) and chooses m nodes in the network to manage. x declares its managee nodes using a $IMAN(node_1, node_2, \cdots, node_m)$ nodes.
5. The nodes which have $node_i$ in their address book can update the list of managers for it.
6. Each node in the network which receives the $IMAN$ message (either from a manager of x or from x itself), adds x with probability $\frac{c}{\sqrt{N}}$ into their address cache and add *all* the managers of x into the table, as they receive the messages from the other managers. Note that this check is done only *once* by each node in the network for a joining node. This ensures that x (and its managers) gets mentioned in expected $c \cdot \sqrt{N}$ address books.

To detect whether a node x is still present in the network or has gone offline, the managers can rely on the blame messages they are receiving. If blame messages for x are being propagated in the network, then x is clearly active and online. If, however, such a message is not received for a time T_1, the managers can ping the node to check its availability. If the node fails to respond for a set number P of pings, it is considered dead. The manager then requests other nodes in the network randomly ($O(\sqrt{N})$ nodes in the average case) to get a list of managers of x and contacts them using unicast messages to get to arrive at a consensus whether x is dead or not. If so, a multicast $REMOVE(x)$ message is sent to the network which directs all nodes which contained an entry for x in their address books to purge this entry.

3.4.2 Comparison with AVMON and AVCast

The two protocols, apart from the described protocol compared here are AVMON and AVCast [10]. Both the protocols use hash function to establish manager/managee relationships. To compare the cost of different protocols, we need a concrete definition of *costs*. For simplicity, costs (with slightly different definitions) in four different scenarios are considered:

1. **Cost of Joining/leaving a system:** In the described protocol, a JOIN message causes $m + 1 \approx O(\log_e N)$ multicast messages, and t messages unicast to the joining node of size $c\sqrt{N} \cdot \log_e N$ each. In AVCast, 1 multicast message (REQ) is sent. For AVMON, the JOIN message is propagated only $O(\sqrt[4]{N})$ times in the network. The side-effect of this gain with respect to JOIN message overhead is the time it takes for a node to be discovered by its manager. While for the described protocol and AVCast, the discovery time is only $O(\frac{\log_e N}{\log_e \log_e N})$[1], for AVMON, it takes $O(\sqrt{N})$ time.
2. **Periodic Costs:** These are the costs for *maintaining* the system, needed primarily in AVMON. Periodically, in AVMON, nodes exchange their *coarse view* of the network, which is of size $O(\sqrt[4]{N})$, and recompute of hash values of $O(\sqrt{N})$ pairs of nodes. The described protocol, however, does not require any periodic updating, while the dependence on periodic updates can be made arbitrarily small in AVCast [5].
3. **Persistent Memory Requirements:** The AVCast protocol in the implementation described in [10] requires $O(N)$ memory. AVMON requires a memory overhead of $O(\sqrt[4]{N})$ and the described protocol, of $O(\log_e N \cdot \sqrt{N})$.
4. **Per-gossip round overhead:** For sending each blame message, the designed protocol would have an overhead of $O(\sqrt{N} \cdot \log_e N)$. For AVMON and AVCast, however, this overhead is surprisingly low, only $O(f \cdot m) \approx O\left((\log_e N)^2\right)$.

Hence, giving up a verifiable relationship between managers and managee nodes results in higher memory as well as bandwidth costs (and lower computation costs). Between AVMON and AVCast, we see that in case of AVCast, the overhead is slightly higher if many nodes join the network, but there are no high periodic costs, while opposite is the case for AVMON. It suggests that in conditions with very high churn, AVMON might perform better than AVCast.

[5] at cost of having slightly *disproportionate* manager/managee relationship

3.5 Conclusions

In this section, the design issues related to a score management system for LiFTinG protocol are looked at. It is seen that the simplest implementation (the central blaming entity) does not scale, while the implementations which does scale introduces a certain structure in the manager/managee relationships, which is undesirable. The middle ground between these two extremes is explored first by keeping managers unknown and gossiping the blame messages to the entire network. This approach was shown to fail on the scalability side. Then a new address caching approach is described and shown to be somewhere between the two approaches. Between AVMON and AVCast too, there exists a trade-off.

Various properties desired in such a protocol were discussed and some basis of comparing the approaches too were shown. Overall, the choice of the protocol to deploy is a complicated question, which can be best answered by the chart shown in Figure 13. *Scalability* refers to the overhead(s) and load balancing of the protocol. *Structure* refers to how *random* is the manager/managee relationship. *Simplicity* is the ease with which the particular algorithms can be implemented (in opinion of the author).

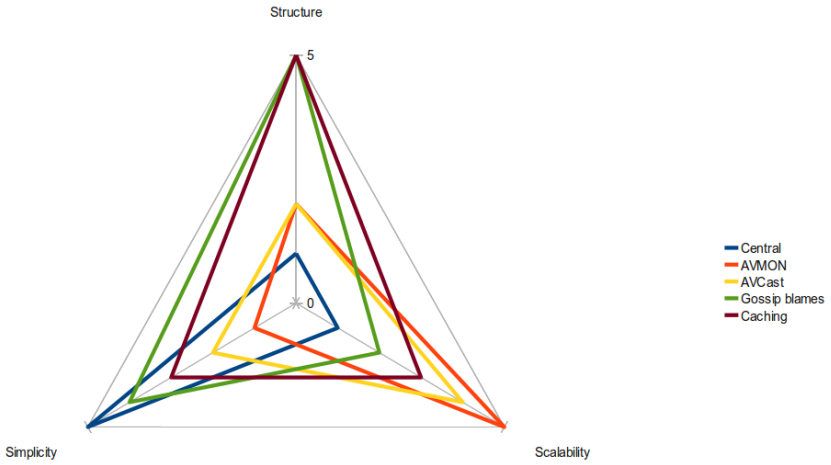


Figure 13: Comparison of various score management schemes discussed in Section 3.4.2 on a scale of 1 to 5, higher score means better on all axes

4 Conclusion & Future work

The first part of the project looked at the problem of interaction between an individual node and a gossiping network with hopes of obtaining an answer to the question how many nodes should *stop* communicating with the individual to keep the individual from obtaining $x\%$ of the data. A model was constructed for the analysis, verified using simulations for practical number of nodes, and then used to predict what the answer would be as N increases.

It was found that this fraction of node increases as the fanout increases in a non-linear fashion (see Figure 9(a)). However, if we turn our perspective around, we see that a node joining the system, to get about 90% of the data, would need to contact a fraction of the total network which keeps on decreasing with number of nodes in the system. While in a system of $N = 50$ nodes, it would need to contact 60% nodes, (or 30 nodes), in a system of $N = 500,000,000$ nodes, it would need to contact 11% of nodes, (or 55,000,000 nodes). This is with the size of the fanout set exactly $\lceil \log_e N \rceil$. If the nodes communicate with a greater fanout, this fraction can be brought down further, without stressing the network much. This investigation is out of scope of this project, but is an interesting direction to work on.

Further, the model which was developed and verified is fairly general and it can be used for many other investigations too. For instance, the model is general enough to predict the behavior of a heterogeneous gossiping-based protocol [3], where the nodes can be divided into segments with different f parameters. Though the temporal evolution as predicted but the model (see Figure 4(b)) has not been verified, it may give insights into the performance of the protocol in terms of jitter too.

The second part of the project looked into the problem of designing an efficient and simple score management protocol. Various faces of the problems were explored and some observations made as to which may be the possible places with some scope of improvement in the present AVMON system, which is *too strong* for our purposes. Also, possible sources of inefficiencies one might encounter while designing such a protocol were also explored and an improved design proposed. However, the analysis revealed that there are numerous hidden parameters which have a role to play in the eventual efficiency of the system. Hence, as a future exploration, a formal description of the *quality* of the ideal protocol could be looked into. This would help in making the pros and cons of different existing protocols clear and would perhaps answer the open question of whether a *better* protocol than AVMON can be used to solve the score management problem in LiFTinG.

5 Acknowledgments

Being a novice in the area of research, this project would not have been possible without constant supervision of Maxime Monod and Prof. Rachid Guerraoui. Prof. Jean-Yves Le Boudec provided much needed guidance on developing the epidemic based model for gossiping and the modeling of outcasts. Prof. Patrick Thiran helped in formulating the *m-coverage* problem and subsequent empirical results. Maxime helped me gain familiarity with the simulator for verifying the models and gave valuable feedback on the report. Also, I thank Rohit Nagpal for intriguing insights and help with the analysis of the models.

Appendix A: Simulation methodology

The simulations for verification of the models were done on the YALP[6] simulator.

The simulations are expensive in terms of CPU, memory, and time taken. Hence, instead of verifying the model for temporal evolution, only the final state of the systems for different parameters was found and was compared to the predictions made by the model.

Verifying the SIR model

To verify the SIR model, the only operative parameter is the $fanout$ or f, and the initial number of sources. The number of sources is not of much interest to us and is assumed to be 1 for all cases in the experiments. Hence, we would verify the system by verifying whether the fraction of nodes reached using a given fanout is the same as what is predicted by the SIR model.

Hence, the ideal way to verify the model would be to run the simulator n times independently for 1 packet each and then observe the fraction of nodes reached by each packet. However, for this verification, the simulation was run for 1000 packets and for 1000 nodes with the fanout ranging from 0 to $\log_e N + 2$. Then the fraction of packets reaching the nodes was calculated and averaged over the number of packets and nodes.

To see how the results of the two different methodologies are related, consider the variables defined in Table 5.

Variable	Quantity denoted
r_i^∞	Fraction of nodes reached by i^{th} packet, $r_i(\infty)$
1_{ij}	Indicator function which is 1 if i^{th} packet reached the j^{th} node
n_j	Number of packets received by j^{th} node.

Table 5: Variables used to analyze simulation results

The quantity we are interested in is $\hat{s}$ or $\hat{r} = 1 - \hat{s}$ for different f, where $\hat{r}$ is given by:

$$\hat{r} = \frac{\sum_{i \in Pckt} r_i^\infty}{\#\ pckts} = \frac{\sum_{i \in Pckts} \sum_{j \in Nodes} 1_{ij}}{\#\ pckts \cdot N} = \frac{\sum_{j \in Nodes} n_j}{\#\ pckts \cdot N}$$

The last is the quantity we measure. Ten repeated runs were performed to get an estimate of the variance, and 97% confidence intervals using the 2^{nd} and the 9^{th} value. This analysis suffices as the variance was found to be negligible, as seen in Figure 4(a). A similar analysis was done for analyzing the model for a fraction of outcast nodes.

Verifying the Outcast modeling

To verify this model, the same simulator was used with the bootstrapping node informing a fraction a of nodes not to contact a given node (or a certain number of nodes). The parameter f was set to $\lceil \log_e n \rceil$. The simulation is then run for the said number of packets and the relevant quantities measured.

To get the probability of infection, the number of packets arriving at the outcast node were divided by the total number of packets gossiped in the network.

[6] http://yalps.gforge.inria.fr/

Appendix B: The $m - coverage$ problem

This section deals with the problem of ensuring that upon performing a random selection of managee nodes, the situation shown in Figure 12 with respect to Node 1 does not happen, that is, all nodes are covered my at least m managers with a high probability. This is framed as a directed random graph problem and analyzed. Thereafter some simulations results are shown and conjectures presented as to what might be the possible solutions. The conclusion is that the number of nodes each node should manage to ensure m managers for each node is conjectured to be $O(\log_e n + f(m))$ where $f(m)$ is $o(m)$.

Problem Statement

Consider a completely connected graph G with n nodes. Each node chooses k other nodes uniformly at random and draws a *directed* edge towards them, thus, resulting in a regular graph of out-degree k. On this graph, we have to find k such that w.h.p[7], each node has an *in-degree* of exactly m. This is referred to as *m-coverage* in the rest of the report.

This corresponds to each node in the network having at least m managers, if the directed arrow between two nodes corresponds to the node at the tail managing the node at the head.

Part of solution

The following simplifications are done to arrive at a different (already solved) problem:

1. The graph is assumed to be $G(n, p_n)$ with $p_n = \frac{k}{n}$. This relaxes the requirement of regularity, or of having exactly k neighbors, to having k neighbors in the average case.
2. The requirement of coverage is replaced with a more stringent requirement of connectivity, noticing that if a network is completely connected, then each node is also covered (except one node, the source)
3. The requirement of having an in-degree of at least m is relaxed to having only at least 1 in-degree.
4. Instead of requiring the probability to be $1 - o(1)$, we require it to be $1 - O(1)$, with the constant being a design parameter.

In this case, the solution is provided in [8]. Which gives us the following result:

If $k = \log_e n + c + o(1)$, then probability of graph being completely connected (covered) is $e^{-e^{-c}}$.

Sketch of the proof

The proof for coverage (on the same lines as proof for connectivity as discussed in [8], but sparing the technical details) is sketched below:

Denote by $\pi_n(m)$ as the probability that in a graph with n nodes, m nodes are covered (have an in-degree of at least 1). Then, we have:

$$\pi_n(n) = 1 - \sum_{i=1}^{i=n} \overbrace{{}^nC_i}^{i \text{ nodes uncovered}} \cdot \underbrace{\pi_n(n-i,k)}_{\text{Prob. rest covered}} \cdot \overbrace{(1-p_n)^{ni}}^{\text{No edge } to \text{ the } i \text{ nodes}}$$

Now if it is assumed that:

1. The sum and limit are interchangeable
2. $\pi_n(n)$ converges to a value π.

then it follows that:

$$\lim_{n\to\infty} {}^nC_i(1-p_n)^{ni} = \frac{n^i \cdot e^{-ki}}{i!}$$

If k is $o(n)$. Now, if we let $k = \log_e n + c + o(1)$, we get:

[7]Defined as probability of event not happening $\approx o(1)$

$$\lim_{n\to\infty} {}^{n}C_i(1-p_n)^{ni} = \frac{e^{-ci}}{i!}$$

Hence, we get the following equation:

$$\pi = 1 - \sum_{i=0}^{i=n} \frac{e^{-ci}}{i!} \cdot \pi$$

On rearrangement, this yields:

$$\pi = e^{-e^{-c}}$$

Hence, proved.

Possible approaches

The easiest would be to find a previous work which deals with a similar problem. However, the following approaches might work too:

1. Somehow modifying the regular pairing graph used to approximate regular graphs to deal with directed edges.
2. Figuring out what is the exact probability of i vertexes having less than m in-degree, when the rest of the graph is m-covered, and then using the same recurrence as used for the relaxed case.

Exploration using simulations

Simulations were performed to check whether k and m followed any easy to see relationships. The results are presented in Table 6.

N	m=1	m=2	m=3	m=5	m=10	m=25
100	10	13	14	16	26	56
373	10	14	16	18	27	56
1,390	12	15	17	20	31	57
5,180	12	15	17	23	33	57
19,307	15	17	20	25	33	57
71,969	16	18	22	25	37	60
268,270	17	20	23	27	37	61
1,000,000	18	22	25	29	40	64

Table 6: Simulation results, 50 repeated runs, probability of coverage > 0.92 with 95% confidence

The evolution of the required k with n can be seen in Figure 14.

Is $k = m \cdot (\log_e n + c)$?

If so, $\frac{k}{m} - \log_e n = c$, or the lines for different m in Figure 15 would be constant lines.
It can, hence, be concluded that k grows much slower than $m \cdot (\log_e n + c)$.

Is $k = m \log_e n + c$?

If so, then $k - m \log_e n = c$, or the lines for different m in Figure 16 would be constant lines.
It can, hence, be concluded that k's dependence on $\log_e n$ is at least sub-linear in m.

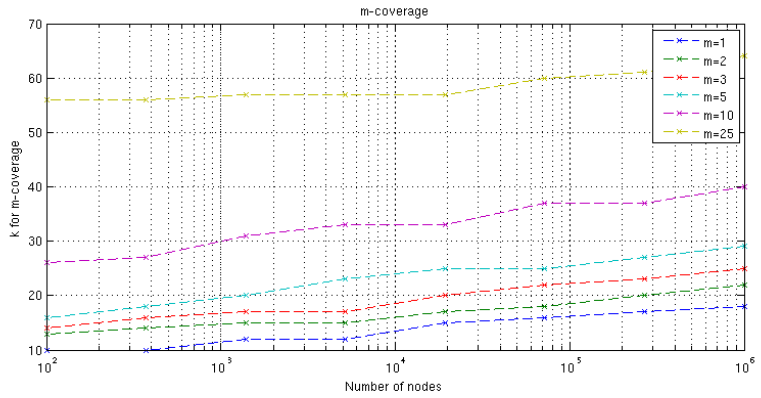


Figure 14: k v/s n for different m coverage

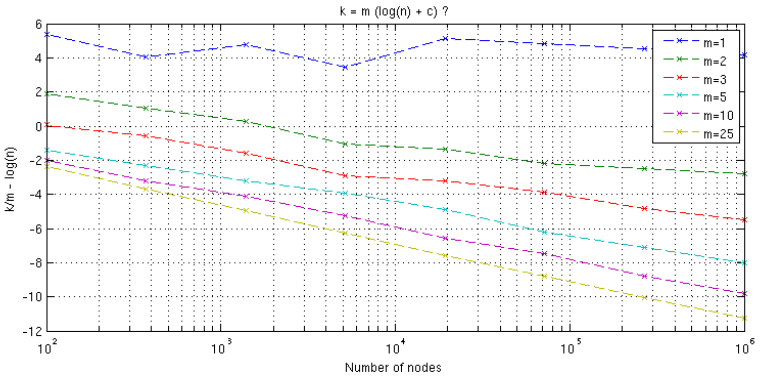


Figure 15: Is $k = m \cdot (\log_e n + c)$?

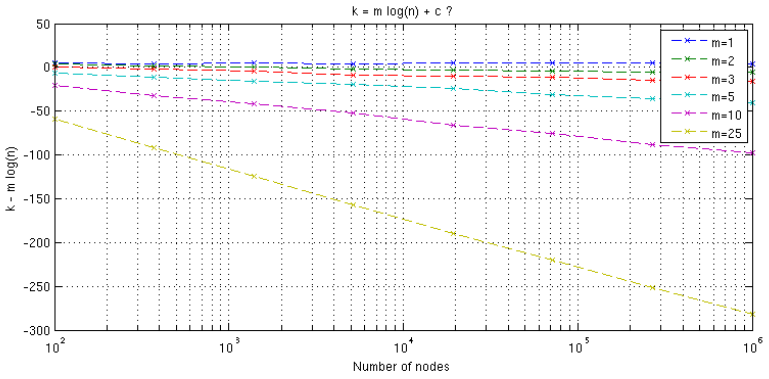


Figure 16: Is $k = m \log_e n + c$?

Is $k = \log_e n + m \cdot c$?

If so, $\frac{k - \log_e n}{m} = c$, or the lines for different m in Figure 17 would be constant lines.

This looks the most promising graph as the lines for high m tend to be constant. However, the decrease in the c with increasing m suggests that the constant c is a function of m and not n. The simulations suggest that $k = \log_e n + f(m)$ is a prospective solution of the problem. However, dependence on $\log_e \log_e n$ or other slowly growing functions, cannot be ruled out.

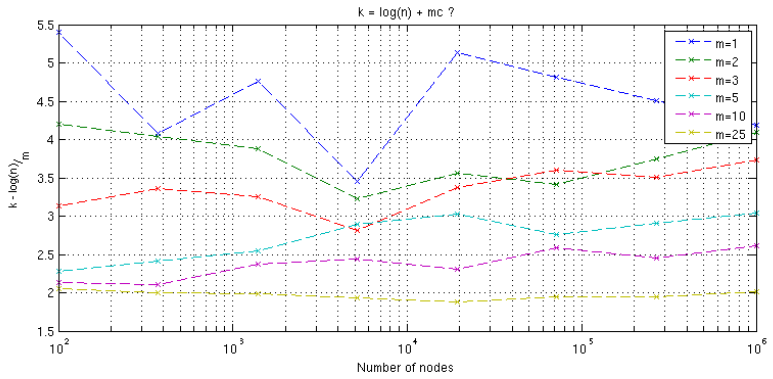


Figure 17: Is $k = \log_e n + m \cdot c$?

References

[1] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[2] P. T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and L. Massoulié. From epidemics to distributed computing. *IEEE Computer*, 37:60–67, 2004.

[3] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Boris Koldehofe, Martin Mogensen, Maxime Monod, and Vivien Quéma. Heterogeneous Gossip. In *Proceedings of the 10th ACM/IFIP/USENIX International Middleware Conference (Middleware)*, 2009.

[4] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, December 1977.

[5] Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. Lifting: Lightweight freerider-tracking protocol in gossip. 2010.

[6] G. Hardin. The tragedy of the commons. *Science*, 20:1243–47, 1968.

[7] Arie Kaufman. A micro-macro simulation model for a signalized network. In *Proceedings of the 13th annual symposium on Simulation (ANSS)*, 1980.

[8] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14:248–258, 2001.

[9] Ramsés Morales and Indranil Gupta. Avmon: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. In *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[10] Thadpong Pongthawornkamol and Indranil Gupta. Avcast : New approaches for implementing availability-dependent reliability for multicast receivers. *IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2006.

[11] Hamidou Tembine, Jean-Yves Le Boudec, Rachid El-Azouzi, and Eitan Altman. Mean Field Asymptotic of Markov Decision Evolutionary Games and Teams. In *Gamenets*, 2009. Invited Paper.