Building a resilient overlay network : Re6stnet

Internship at Nexedi KK

Ulysse Beaugnon

September 5, 2012

Contents

1 Introduction

Nexedi provides services from servers hosted in unconventional places like private houses in addition to traditional data-centers. They make extensive use of IPv6 (they need many IPs on a single machine for virtualization) but the current IPv6 support is very bad. Many houses have no access to it and native IPv6 networks suffer from a lot of problems [10]. Using native IPv6 in houses is impossible for any professional usage.

The common workaround is to create tunnels from each peer to a server with a good IPv6 connection. This solution doesn't scale well and is not resilient at all. It also requires to have multiple servers around the world to reduce the latency overhead. The goal of my internship was to build a scalable, decentralized and resilient peer-to-peer overlay network by creating tunnels between peers to answer those problems. I have focused on establishing tunnels in a resilient, efficient and scalable way and not on routing.

Peer-to-peers overlay networks improve reliability of connections, even for IPv4 [16]. Since we control routing tables, we recover faster from a link failure than BGP or other algorithms used by Internet providers. This also improves latency : because of routing policies, the path chosen is not always the shortest and the triangular inequality is not respected among peers. Using a detour route reduces latency [21].

I was working on this project with two colleagues : another intern (Guillaume Bury) and our supervisor (Julien Muchembled). I was in Tokyo while they were working from France. Since we had seven hours of time difference, I used to work on my own all the day and explain to Julien what I had done and what I had planned to do next before leaving the office.

I have mostly focused on finding an algorithm establishing the tunnels between peers while Guillaume has focused on the implementation choices and choosing the external routing algorithm we use to route over our network. Julien was only here to check what we had done, give us some advices and answer our questions. We have both worked on the implementation of our algorithm.

I have started my internship by requesting a /48 IPv6 subnet to the RIPE to address our network. Then, I have tried to find an algorithm to establish tunnels without considering latency. For this, I first tried to find the best graphs possible in term of resilience and distance we could do. Then I looked at the existing algorithms to build overlay networks, for example in distributed has tables before choosing a random graph approach. To check the performances of our algorithm, I have used simulations as well as heuristics and mathematical bounds on the probability for the network to end-up disconnected.

Once we had a working random network, I have considered performances in term of latency and worked on finding an algorithm to incrementally optimize our network by keeping only the best tunnels and replacing the others. This was only based on simulation. At the end, I have been working on removing the need for a central point in our overlay network.

2 The optimum

Since every peer can host a limited number of tunnels, we have only focused on k-regular undirected graphs. We use an external routing algorithm to route over our network.

2.1 Optimal distance

To avoid catastrophic performances, we must be careful to the number of hops between nodes. The Moore bound gives a limit on the number of nodes n for a k-regular graph of diameter D [7] :

$$n \leq \sum_{i=0}^{D} k^i \quad \Leftrightarrow \quad D \geq D_m = \lceil \log_k (n(k-1)+1) \rceil - 1 \tag{2.1}$$

[17] also gives a lower bound on the average distance d for a k-regular graph with n nodes :

$$d \geq D_m - \frac{k^{D_m+1} - k \cdot l - d + D_m}{n(k-1)^2} \tag{2.2}$$

This bounds are not reachable for non-trivial cases [22] but De Bruijn graphs can get close to them [13] [15].

2.2 Optimal resilience

Since we rely on peers we don't control, we have to assume they may fail or leave the network at any moment [20]. An attacker might also try to kill selected peers to damage our network. Thus we have to consider resilience to avoid loosing connectivity.

A way to measure resilience is to count the number of nodes or edges one must remove to disconnect two nodes in the graph. Since each node has k neighbors, the optimum is to have k paths with no common nodes except the first and the last one between each pair of peers. Modified De Bruijn graphs can achieve this constraints without altering their almost optimal average distance and diameter [8][9].

3 A random network

3.1 Building a network without any global information

Having a complete view of the network on each peer is impossible for a scalable network. Peers might often leave and enter it. Broadcasting information about their status each time would saturate the links.

Structured overlay network, used in distributed hash tables like Chord or Kademlia [12][18], are able to build a network with $O(\ln(n))$ information on each node. But this still require some overhead and links are established according to a fixed topology that doesn't take into account the underlying network parameters. This is OK when we don't care about latency but in our case it might lead to bad topologies.

Since no pertinent information can be collected without too much overhead, we have used random graphs to generate our network. The only information a peer needs is the external IP and port of a few randomly chosen peers to be able to connect to them.

Random k-regular graphs have good distances and resilience proprieties. They are almost surely k-connected [5] and according to [4] their diameter is almost surely below :

$$\left\lceil \log_{k-1} (k \cdot n) + \log_{k-1} ((2+\epsilon) \log n) \right\rceil + 1 \tag{3.1}$$

They are almost optimal both in term of resilience and distance compared to the bound given in 2.1.

3.2 Our algorithm

Random k-regular graphs are difficult to generate. A lot of coordination between nodes are required to choose the tunnels to establish. Instead of that, we generate a graph where the arity of a node is $\geq k$ and close to $2 \cdot k$ with high probability.

Each peer rely on a local list of possible nodes to connect to (the local DB) to establish a tunnel to k randomly chosen peers. Since each peer creates k tunnels, a node has in average $2 \cdot k$ tunnels. The arity is distributed according to the following formula :

$$\mathbf{P}\left(arity = a + k\right) = \binom{n-1}{a} \left(\frac{k}{n-1}\right)^a \left(1 - \frac{k}{n-1}\right)^{n-a-1} \tag{3.2}$$

With $k = 10$ and $n = 1000$, 99.9% of the nodes have an arity between 12 and 32. Thus, we can reasonably hope that our random network will have properties close to random regular graphs.

If a tunnel is detected to be down, it is immediately replaced by a new randomly chosen one. Since a peer only cares about the tunnels it has established, no cooperation is needed among nodes. To include new peers into the network, each peer replace a randomly chosen tunnel among the ones it has established every t seconds.

To avoid that two nodes both establish a tunnel to the other, a node reminds peers connected to it. If two nodes both establish a tunnel to the other at the same time, they compare their SSL certificate serial number to chose which tunnel to keep.

Each peer regularly sends the necessary information to connect to it (external IP, port, ...) to a server : the registry. The registry reminds all the peers that have contacted him recently (so dead peers won't stay in its database). When a peer needs to refresh its local DB, it asks the registry for a list of peers. The registry is also used to generate the SSL certificates necessary to enter the network and attribute static IPs to peers.

3.3 Advantages

The only information stored on each peer is a partial list of other peers of constant size and no information has to be exchanged between peers. They only need to contact the registry a few times a day. Thus, the overhead generated by the maintenance of the network is very small and the network is scalable.

Except for the registry, the network is decentralized. The registry can fail for a short time since every peer has a local DB and can survive alone as long as it is not empty. If the registry is not reachable from a peer (for example if it is censored in a particular country), the local DB can be initially filled by hand and then the registry reached through the overlay network. This makes our network resilient.

4 Performances of the random network

4.1 Chances that the algorithm fails

In our algorithm, there is no guarantee that the network is connected. Two nodes can be disconnected even if no other node or tunnel has failed. An upper bound $\hat{P}_d$ can be given on the probability P_d of such a catastrophic failure :

$$P_d \leq \hat{P}_d = \sum_{m=2k+1}^{n-2k+1} \binom{n-1}{m-1} \frac{\binom{n-m-1}{k}^{n-m} \binom{m-1}{k}^{m}}{\binom{n-1}{k}^{n}} \tag{4.1}$$

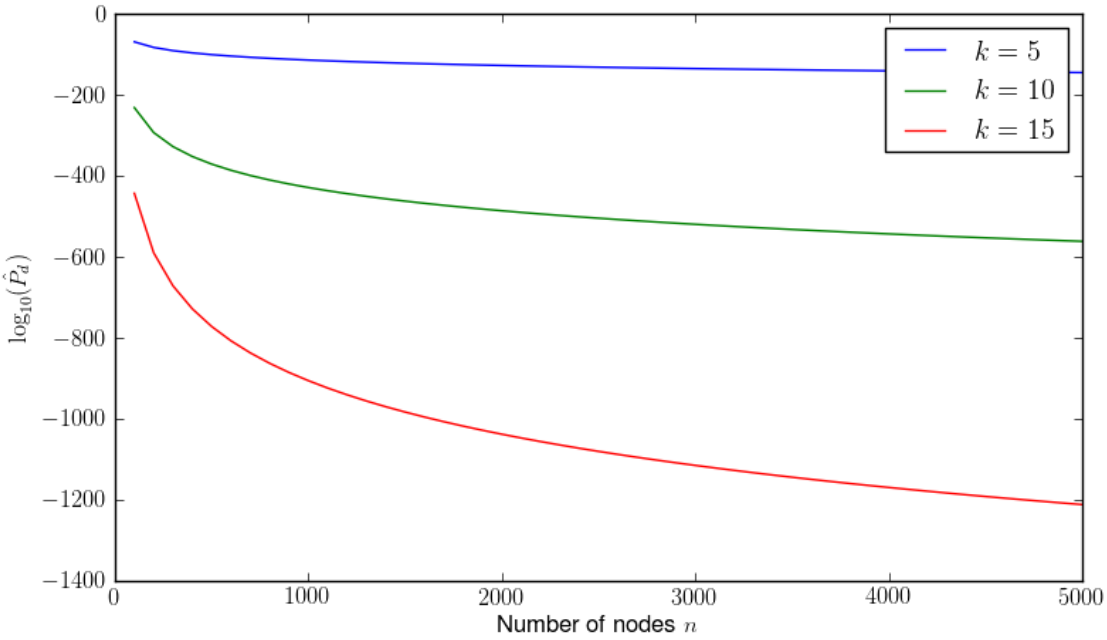

Figure 1: Chances that the algorithm fails

As shown on figure 1, this probability is very low and catastrophic failures should not be a problem.

Proof of (4.1). :
Let $G(V,\ E)$ be a labeled graph generated by our algorithm. Let $n = |V|$, $v \in V$ and A the maximal connected component of v. If $X, Y \in V$, $X \perp Y$ means there is no edge between X and Y.

$$\begin{aligned} P_d &= \mathbf{P}\left(G \text{ is not connected}\right) && (4.2)\\ &= \mathbf{P}\left(A \subsetneq V\right) && (4.3)\\ &= \sum_{\substack{B \subsetneq V \\ v \in B}} \mathbf{P}\left(B \text{ is a maximum connected componnent}\right) && (4.4)\\ &\leq \sum_{\substack{B \subsetneq V \\ v \in B}} \mathbf{P}\left(B \perp V_{\setminus B}\right) && (4.5)\\ &\leq \sum_{m=1}^{n-1} \sum_{\substack{|B|=m \\ v \in B}} \mathbf{P}\left(B \perp V_{\setminus B}\right) && (4.6) \end{aligned}$$

Let $B \subsetneq V$ such that $B \perp V_{\setminus B}$ and $m = |B|$. Since each vertice of B generates k edge and that there is $\frac{m \cdot (m-1)}{2}$ possible edge in B :

$$\begin{aligned} m \cdot k &\leq \frac{m \cdot (m-1)}{2} && (4.7)\\ \Leftrightarrow \quad m &\leq 2 \cdot k + 1 && (4.8) \end{aligned}$$

(4.8) also stand for $m = \left|V_{\setminus B}\right|$, thus :

$$2 \cdot k + 1 \leq m \leq n - (2 \cdot k + 1) \tag{4.9}$$

Then :

$$P_d \leq \sum_{m=2\cdot k+1}^{n-2\cdot k+1} \sum_{\substack{|B|=m \\ v\in B}} \mathbf{P}\left(B \perp V_{\setminus B}\right) \tag{4.10}$$

Let $B \subset V$ such that $v \in B$, and $|B| = m$. In our algorithm, when 2 nodes both establish a tunnel to the other, one of the tunnels is replaced. This can only decrease $\mathbf{P}\left(B \perp V_{\setminus B}\right)$ since the replaced tunnel was useless. As we only want an upper bound, we can do as if tow nodes could make tunnels to each other when calculating $\mathbf{P}\left(B \perp V_{\setminus B}\right)$.

When we generate the graph, each node has $\binom{n-1}{k}$ possibilities to create its tunnels. Thus, the number of possible E is $\binom{n-1}{k}^n$. If we impose that $B \perp V_{\setminus B}$, then each node of B has $\binom{m-1}{k}$ possibilities and each node of $V_{\setminus B}$ and $\binom{n-m-1}{k}$ possibilities. Thus the number of possible E when $B \perp V_{\setminus B}$ is $\binom{n-m-1}{k}^{n-m}\binom{m-1}{k}^m$. Then:

$$\mathbf{P}\left(B \perp V_{\setminus B}\right) \leq \frac{\text{number of possible } E \text{ were } B \perp V_{\setminus B}}{\text{number of possible } E} \tag{4.11}$$

$$\leq \frac{\binom{n-m-1}{k}^{n-m}\binom{m-1}{k}^m}{\binom{n-1}{k}^n} \tag{4.12}$$

This and (4.10) give us :

$$P_d \leq \sum_{m=2\cdot k+1}^{n-2\cdot k+1} \sum_{\substack{|B|=m \\ v\in B}} \frac{\binom{n-m-1}{k}^{n-m}\binom{m-1}{k}^m}{\binom{n-1}{k}^n} \tag{4.13}$$

$$\leq \sum_{m=2\cdot k+1}^{n-2\cdot k+1} \binom{n-1}{m-1} \frac{\binom{n-m-1}{k}^{n-m}\binom{m-1}{k}^m}{\binom{n-1}{k}^n} \tag{4.14}$$

$$\leq \hat{P}_d \tag{4.15}$$

□

4.2 Average distance

We give an heuristic on the number u_d of node at a distance d of a given node :

$$\begin{cases} u_0 = 1 \\ u_1 = 2k \\ u_{d+1} = \left(1 - \sum_{i=0}^{d} \frac{u_i}{n}\right) \cdot (2k-1)u_d \qquad \forall d \geq 1 \end{cases} \tag{4.16}$$

This allows us to give an heuristic on the average distance $\bar{D}$. If $d \in \mathbb{N}^*$ such that $u_{d-1} \leq n < u_d$, then :

$$\bar{D} \approx \sum_{i=0}^{d-1} i \cdot u_i + d \cdot (n - \sum_{i=0}^{d-1} \cdot u_i) \tag{4.17}$$

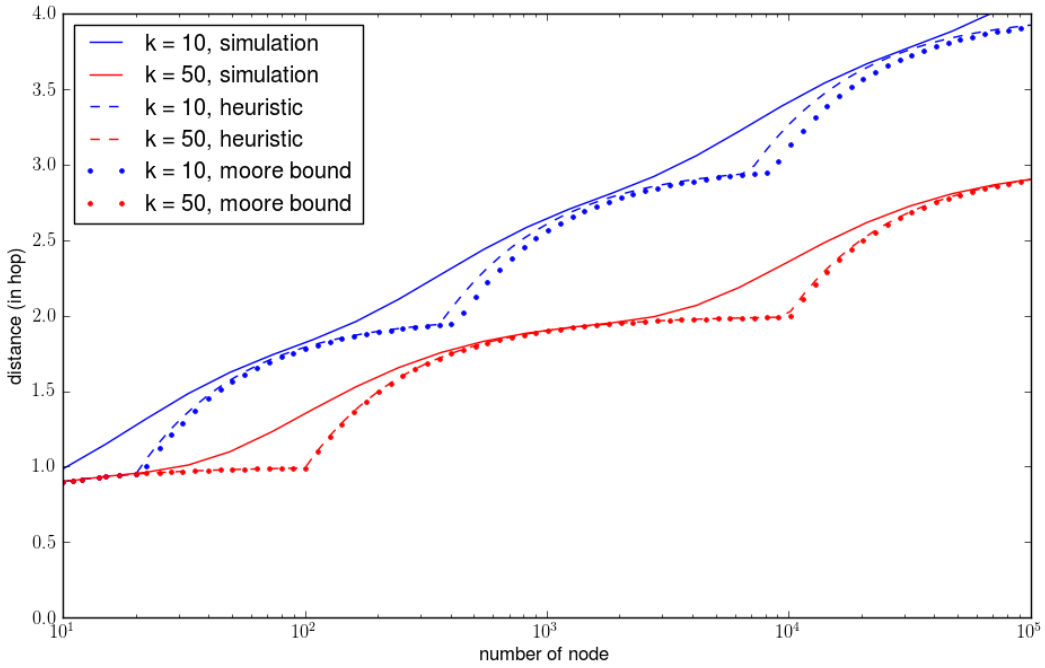

Figure 2: Average distance (simulation and heuristic)

As shown on figure 2, the heuristic follows the simulation, especially when k is high. The distance is also close to the optimum given in (2.2) for a constant number of tunnels of $2 \cdot k$.

Origin of (4.16). :
Let v be a node of our network. We note V_d the nodes at distance d of v and $u_d = |Q|$. Each node having on average $2k$ tunnels (establish by itself and other nodes), we assume in the heuristic that each node has exactly $2k$ tunnels. Thus :

$$\begin{cases} u_0 &= 1 \\ u_1 &= 2k \end{cases} \tag{4.18}$$

$u_{d+1} \approx$ the number of tunnels from Q_d which doesn't link to $\bigcup_{i \leq d} Q_i$. For $d \geq 1$, each node of V_d has one tunnel linking it to V_{d-1}. This leaves $2k-1$ potential tunnels to connect to Q_{d+1}. A new tunnel has a probability $\sum_{i=0}^{d} \frac{u_i}{n}$ of connecting to $\bigcup_{i \leq d} Q_i$. Then approximatively $\left(1 - \sum_{i=0}^{d} \frac{u_i}{n}\right) \cdot (2k-1)$ tunnels from Q_d link to Q_{d+1}. This gives us :

$$u_{d+1} = \left(1 - \sum_{i=0}^{d} \frac{u_i}{n}\right) \cdot (2k-1) u_d \tag{4.19}$$

Which is exactly (4.16). □

4.3 Fault tolerance

There is a bound $\hat{P}_u$ on the probability that two node can't reach each other after q nodes randomly chosen have failed.

$$P_u \leq \hat{P}_u = \sum_{m=1}^{n-q-1} \binom{n-2-q}{m-1} \frac{\binom{n-m-1}{k}^{n-m-q} \binom{m+q-1}{k}^{m}}{\binom{n-1}{k}^{n-q}} \tag{4.20}$$

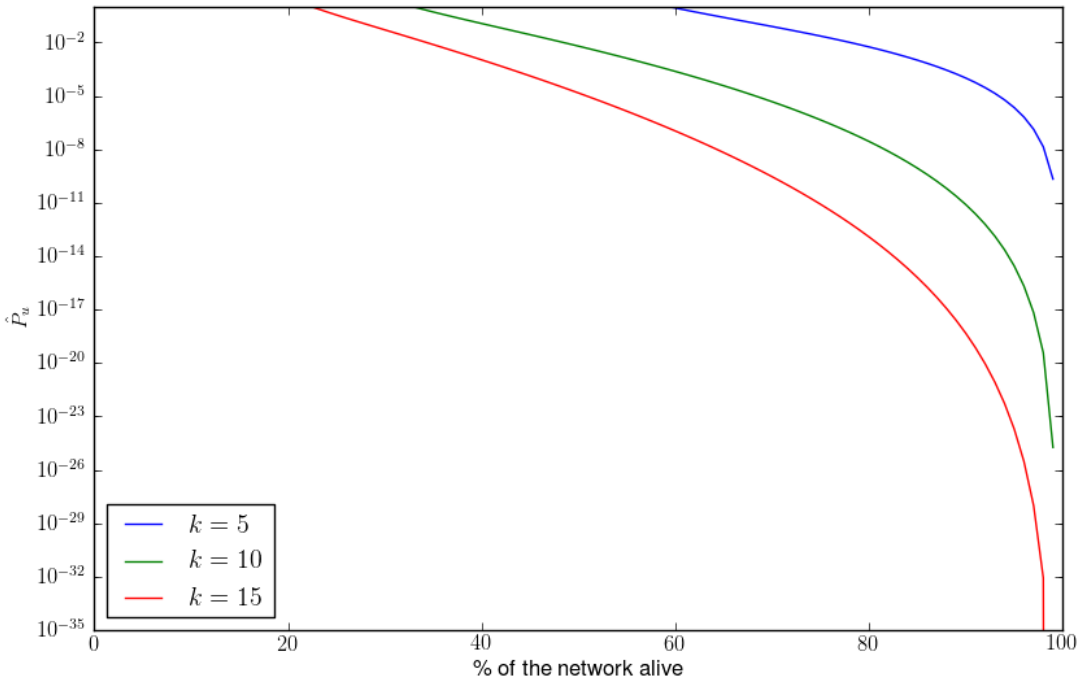

Figure 3: bound (4.20) for $n = 1000$

We also made a simulation where we built a random graph following our algorithm and then removed a portion of the peers. As shown on figure 4, results from the simulation are much better than the bound (4.20) when a big portion of the network is dead.

Proof of (4.20). :
Let $G(V,\ E)$ a graph generated by our algorithm, $v_0,\ v_1 \in V$, $Q \subsetneq V_{\setminus\{v_0,\, v_1\}}$ and $q = |Q|$. We want the probability P_u that v_1 is unreachable from v_0 when all the nodes of Q are dead. As for (4.12), we suppose that the graph was generated without replacing tunnels when 2 nodes both connect to the other.

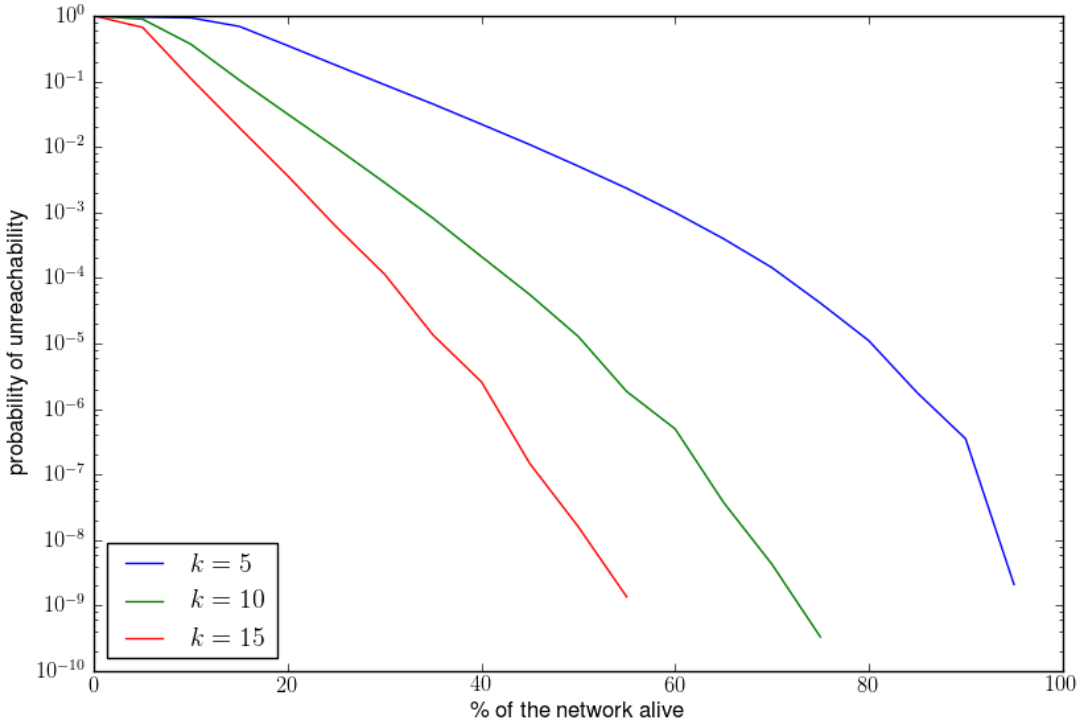

Figure 4: Simulation of a simultaneous failure on several nodes for $n = 1000$

$$
\begin{aligned}
P_u &= \mathbf{P}\left(\text{The maximal connected component of } v_0 \text{ doesn't contain } v_1\right) && (4.21)\\
&= \sum_{\substack{A \subsetneq V_{\setminus\{v_1\}} \\ v_0 \in A}} \mathbf{P}\left(A \text{ is a maximal connected component}\right) && (4.22)\\
&\leq \sum_{\substack{A \subsetneq V_{\setminus\{v_1\}} \\ v_0 \in A}} \mathbf{P}\left(A \perp V_{\setminus A \cup Q}\right) && (4.23)\\
&\leq \sum_{m=1}^{n-q-1} \sum_{\substack{A \subsetneq V_{\setminus\{v_1\}} \\ |A|=m \\ v_0 \in A}} \mathbf{P}\left(A \perp V_{\setminus A \cup Q}\right) && (4.24)\\
&\leq \sum_{m=1}^{n-q-1} \sum_{\substack{A \subsetneq V_{\setminus\{v_1\}} \\ |A|=m \\ v_0 \in A}} \frac{\text{number of possible } E \text{ where } A \perp V_{\setminus A \cup Q}}{\text{number of possible } E} && (4.25)
\end{aligned}
$$

As for (4.12), the number of possible E is $\binom{n-1}{k}^n$. When $A \perp V_{\setminus A \cup Q}$, a node in Q can connect to any other node, a node in A can connect to any other node in $A \cup Q$ and a node in $V_{\setminus A \cup Q}$ can connect to any other node in $V_{\setminus A}$. This makes $\binom{m+q-1}{k}^m \binom{n-1}{k}^q \binom{n-m-1}{k}^{n-m-q}$ possibilities for E. Then :

$$P_u \leq \sum_{m=1}^{n-q-1} \sum_{\substack{A \subsetneq V \setminus \{v_1\} \\ |A|=m \\ v_0 \in A}} \frac{\binom{m+q-1}{k}^m \binom{n-m-1}{k}^{n-m-q}}{\binom{n-1}{k}^{n-q}} \tag{4.26}$$

$$\leq \sum_{m=1}^{n-q-1} \binom{n-2-q}{m-1} \frac{\binom{m+q-1}{k}^m \binom{n-m-1}{k}^{n-m-q}}{\binom{n-1}{k}^{n-q}} \tag{4.27}$$

$$\leq \hat{P}_u \tag{4.28}$$

□

4.4 Availability

Ensuring that the network stays connected is essential but even if it does, the routing algorithm will need a little time to detect a failure and establish a new route. This result in a small unavailability for routes.

$$\text{unavailability} = \frac{\text{average distance} \cdot \text{recover time}}{\text{mean time between failure of a node}} \tag{4.29}$$

With the routing algorithm the recover time is typically 1 minute and for Nexedi's servers, the mean time between failure is 3.6 days. According to figure 4.16, with $k = 10$, this gives us an unavailability of $5.0 \cdot 10^{-4}$ for 10^3 nodes and of $9.4 \cdot 10^{-4}$ for 10^6 nodes.

4.5 Choosing the number of tunnels to establish

In our network, each node establish 10 tunnels ($k = 10$). This allows us to have a reasonable distance and resilience while the overhead due to hosting tunnels remains small. As shown on figure 5, the improvement in distance is much smaller after $k = 10$.

To avoid having too many tunnels on the same node, we have also set a maximal arity of $3 \cdot k$. If a node tries to connect to a node which is already full, then the connection will be refused and the node will try to connect to someone else. This shouldn't change a lot of thing since it only affects a small number of peers (0.3% of nodes with 1000 nodes). In our simulations no effect was visible.

4.6 Taking latency into account

The measure of distance in hop gives a good idea of how things are but some tunnels might have a much bigger latency than others. Computing the shortest path using latency is necessary to achieve good performances. For example, in simulations on our dataset, when routing in number of hop, the average latency was 220 ms while it was 90 ms when taking it into account.

To make simulations with latency, we have used a dataset [1] giving RTTs between 2500 nodes. The average latency when using only direct connection was 75.8 ms and 16.1 ms with a full overlay mesh network making the use of detour routes possible.

5 Optimizing the randomized network

5.1 Evaluate the usefulness of tunnels

Even if our network has good performances, it can still be improved since some tunnels are almost useless and could be replaced by more useful ones. For example, if a node has a really bad connection no traffic will

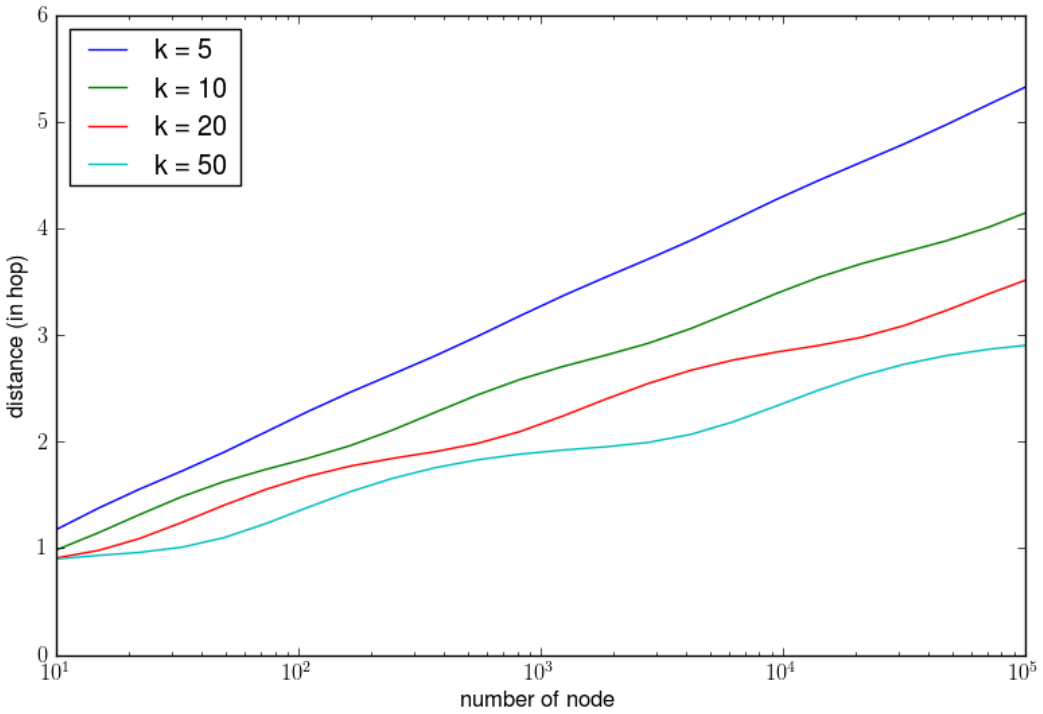

Figure 5: Average distance

transit through it and it is useless for other to connect to it. Tunnels established by the node itself should be sufficient to ensure its good connectivity.

As explained in 3.1, it is difficult to choose the best tunnel to create without a lot of informations on the network. But once the tunnel is establish, it is possible to make local measures on it to estimate its usefulness. Instead of randomly choosing a tunnel to replace every t minutes, we choose the less useful one. After many iterations, only the most useful will remain : we incrementally optimize our network.

The most obvious way to measure a tunnel usefulness is to look at the amount of traffic going through it. This can be done very easily using the kernel stats. Favor such tunnel will favor the most used routes and tends to make faster connections between nodes that needs to speak a lot to each other. This can seam to be a good idea but it also raise some problems. Firstly, if a group of nodes talk a lot together and never to the other nodes, they will end disconnected up from the rest. Secondly, it brings some instabilities in the network since the most used routes are not always the same.

An other way is to try to keep tunnels with a good latency. Favor those tunnels will avoid the bad ones but a node will tends to stay connected only to close peers and a group of well connected together machines will eventually end up disconnected from the rest.

A last way to evaluate the usefulness of a tunnel is to count the number of peers reached through it. This information is retrieved from the routing table. This is the measure we have used to choose which tunnel we should replace at every iteration of the algorithm.

5.2 Effects of the optimization

Replacing tunnels with the less peers reached tends to sort peers in tow equally-sized categories : nodes with an almost maximal arity and nodes with an almost minimal arity (figure 6).

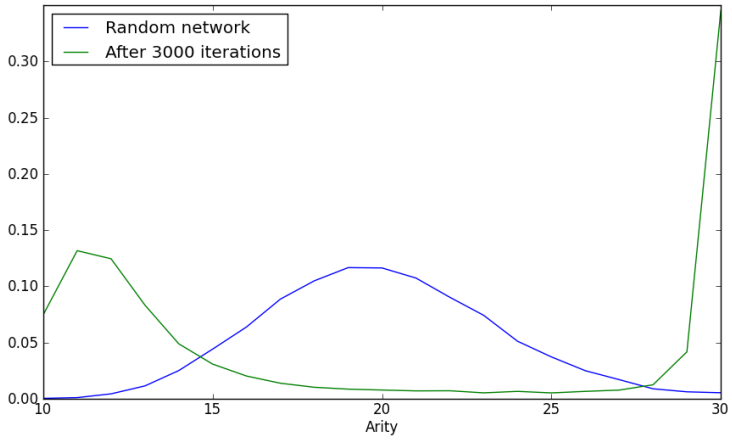

Figure 6: Evolution of the arity distribution

Nodes with an almost maximal arity are the more central nodes in the overlay network. They allow peers connected to them reach a lot of other peers so others stay connected to them. On the other side, less central nodes have an almost minimal arity : others have dropped tunnels they had established to them. Since a high arity increase the centrality, central nodes tends to become even more central. This is why the tow categories of peers are so well separated.

Peers with a low average latency to others on the underlying network have a higher chance of becoming central as this can be seen on figure 7. More reliable peers are also advantaged since they have a longer life and so, a bigger chance of becoming central. This can be observed on figure 8 where we have given to each peer a probability p of leaving and instantaneously re-enter the network between tow iterations as if the node had been shutdown or disconnected from the Internet for a short time. Thus our algorithm automatically choose the most well-connected and reliable peers to make them a little more important and reduce latency.

The average latency between nodes is greatly improved by this incremental optimization. As shown on figure 9, the average latency fall from $\approx$ 90 ms to $\approx$ 20 ms. This is much better than the average latency through direct connections ($\approx$ 75 ms) and close to the full mesh network ($\approx$ 16 ms).

Many iteration have to be made before the average latency is almost optimal. Since each iteration generates some traffic to update routes, we can't increase their frequency too much. This is OK when the network is stable since it will ultimately reach almost optimal performances, but when peers often leave and enter, performances are decreased : the almost optimal topology has no time to be established. For example, if each peer leave and instantaneously re-enter the network with a probability $\frac{1}{100}$ at each iteration, the average latency was 50 ms in our simulation. This is still better than direct connections but not as good that what is achievable with a stable network.

Since each node still have at least 10 tunnels and that not only one or tow but half of the peers are more important, we still have a mesh network and we keep a reasonable resilience (figure 10).

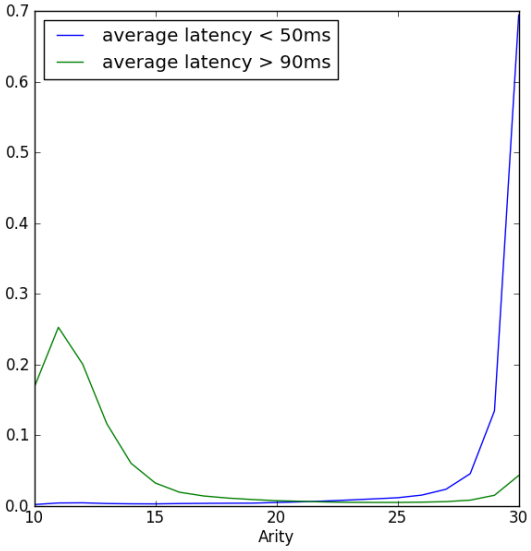

Figure 7: Distribution of the arity depending on the average latency of the direct connections to the other nodes

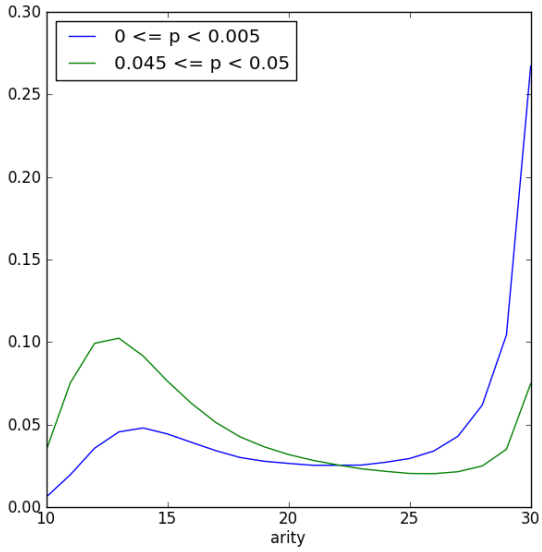

Figure 8: Arity depending on reliability

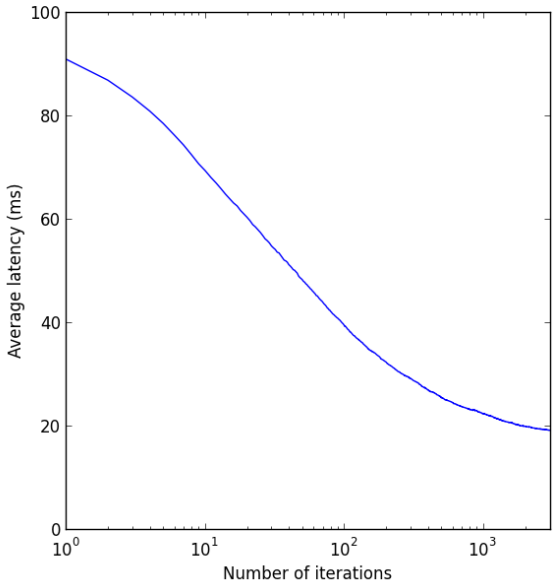

Figure 9: Evolution of the average latency

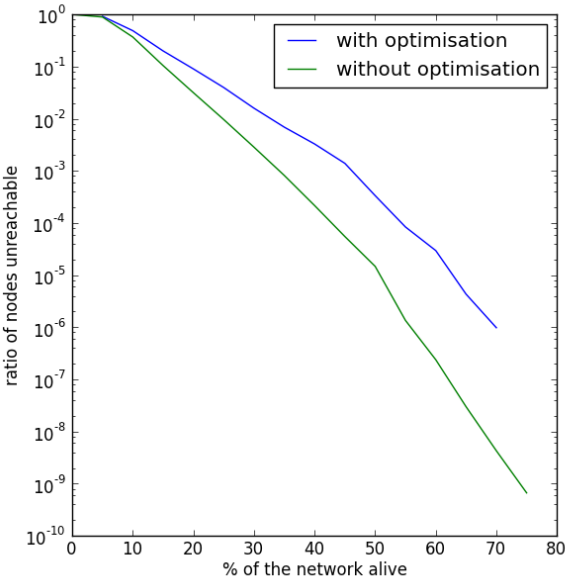

Figure 10: Simulation of a simultaneous failure of several nodes

Replacing only tunnels with the less peer reached also reduce the traffic generated by the routing algorithm since less routes are changed. In our dataset (with 2500 nodes), the average number of peers reached by the tunnels replaced was 1.85 while the average is 75 for a tunnel.

6 Reducing the need for a registry

Our algorithm use the registry as a central point to exchange informations. If it is down for a long time, peers cannot discover each others anymore. Its neighbor might also have to forward a lot of traffic if the network grows too big. Thus, the registry should only be used when a peer knows no other peers (for example when it first enter the network) so there is no central point of failure and the network will scale better.

To allow others to connect to it, a peer randomly choose a few addresses belonging to the network in its routing table and send them its external informations (external IP, port, ...) so the receivers can store them in their local DB. A new peers has to send its external information to a number of node equal to the size of the local DB so it has as many chances as other to be chosen for establishing a tunnel.

Even if nodes can fill their DB by themselves, the registry is still needed to ensure the network is connected and when a peer knows no other peer. Twice a day a node contact the registry to give him its external informations. If it can't reach the registry through the overlay network, then it contact it by its public IP to ask for new peers as if it was first entering the network. This way, if a set of nodes get disconnected, it will be reconnected as soon as one of them tries to recontact the server.

7 Our implementation

Our implementation is based on a python script to contact the registry and manage tunnels. The script launch an OpenVPN server and several OpenVPN client [2] to establish the tunnels. It also launch the routing daemon. The script is able to automatically ask for a port forward to a router using UPnP-IGD protocol.

In order to communicate with the Internet through our network we have requested a /48 IPv6 subnet so IPs given to nodes will be routable on the Internet and can be accessed through a gateway making the link between the real IPv6 network and the overlay network.

Our implementation also detect when some other peers are available in the LAN to avoid establishing tunnels with them and instead use a direct connection.

All features have been tested using Marionnet which permit to build a virtual network between virtual machines [14].

To route over our network we use the Babel routing protocol [6]. It was designed specifically for mesh networks and can recover fast from a node failure. The latency measurement has not been implemented in it yet but will soon be.

8 Future improvements

8.1 Hierarchical routing

There is currently no hierarchical routing in our network. Because of this, the routing table size and the routing information exchanged between each neighbors is in $O(n)$. Implementing a hierarchical routing is

essential if we want our network to be scalable since it would reduce the size of the routing table and the information exchanged to $O(\ln(n))$.

Usually, subnets are assigned statically and manually at the creation of a network. But here, when a node ask to join, we have no information about it nor about the topology of the network so it is impossible to choose in which subnet we should put it. And even if we manage to do so, the network might evolve in a very different way. Moreover, since nodes from a single subnet must stay connected, having a static addressing would add a constraint and reduce resilience. Thus a dynamic addressing able to follow the evolutions of the network and to recover from a failure is mandatory.

Even if the addressing is dynamic, some services still need to have a static IP. To achieve this each peer must run a NAT and have both a static and a dynamic IP. A packet is sent to an other machine using its static IP. The NAT of the emitter changes the static IP of the destination to its dynamic IPs and the NAT of the receiver change it back to the static IP. This way, the overlay network only sees dynamic IPs and machines only see static IPs. The index necessary to store the indirection between static and dynamic IPs can be stored in a DHT with a cache on each peer. When a node changes its IP, it can keep both the old and the new one until the information has propagated.

Multiple algorithms and their implementations exist to automatically build a hierarchical mesh network [19], but some work might be needed to adapt them to our network since most of them have been developed for specific environments like ad hoc wireless networks.

Every subnet will contain ≈ 100 machines (or smaller subnet). Inside a subnet the routing will be optimal and outside a subnet, a packet will be routed on the shortest path to enter to its subnet (and not on the shortest path to its destination). Since the routing will not be optimal, latency will be affected if the number of machine in a subnet is big enough, the effect should be small.

The parts of our algorithm relying on the routing table (described in 5.1 and 6) will have to be adapted to work with a hierarchical routing. If for each subnet in its routing table, a peer knows the number of peers it contains, then it can count the number of peers accessed through each tunnel. It can also send a message to an uniformly chosen peer : it choose a subnet (or peer) S in its routing table with a probability proportional to the number of peers inside S. If S is a machine, it only need to sends the packet to it. If S is a subnet, it anycast the packet to this subnet. When the packet is received by a node of S it is forwarded to a randomly chosen machine inside S using the same technique recursively.

The number of nodes in a subnet can be transmitted at the same time than routes. A node starts by advertising its own subnet saying there is one node in it. Then, whenever a node receive a route to a subnet, it knows the number of nodes inside it. When a node aggregate some routes, it just has to sum up the number of nodes in the routes aggregated.

8.2 Security concerns

We currently control who is connecting to our network using OpenSSL certificates for OpenVPN. But if an evil peer manage to get a certificate, it may damage the network. Some security issues have to be solved to answer this problem.

The first thing necessary is the possibility to revoke some certificates. This could be achieved by storing the revoked certificates in a server contacted once a day by peers to update their list of revoked certificates. An algorithm to detect peers behaving badly (for example a node refusing to forward traffic) would also be necessary.

We also need to ensure that external node cannot send false information about entry-points inside the network using a peer with a badly configured firewall or that a single node cannot advertise many false entry-point. This can be done by signing every message sent for entry-point advertising. This way, we can check that the message was sent by someone with a certificate granted by the registry and that it only advertise one entry-point.

The last problem is that a node can send false routing information. This problem is not limited to our network and the Internet face the same problem [11].

9 Conclusion

We have presented a way to build a dynamic peer-to-peer unstructured overlay network. Our algorithm is inspired from the good properties of random regular graphs. It can be used to provide a more reliable and faster connection since it can use detour routes. Our algorithm was validated using heuristics, bounds and simulations.

We then present a way to optimize the random network to get lower latency while keeping an acceptable resilience. With this optimization, on our dataset, we could achieve a better latency than direct connections thanks to detour routes.

Once hierarchical routing will be implemented, our network should scale well since their is no structure or global information to maintain except the routing table and a list of peers on each node. Unlike some other overlay network like RON [3] trying to improve resilience and latency, the number of neighbor is bounded.

Our algorithm will be used by Nexedi to provide a fast and reliable IPv6 to its servers.

References

[1] The meridian dataset. `http://www.cs.cornell.edu/People/egs/meridian/data.php`.

[2] Openvpn website. `http://openvpn.net/`.

[3] David G. Andersen. Resilient overlay networks. *Massachusetts Institute of Thechnology*, 2001.

[4] W. Fernandez de la Vega B. Bollobas. The diameter of random regular graphs. *Combinatorica*, 1981.

[5] Bela Bollobas. Random graphs. *Cambridge University Press*, 2001.

[6] J. Chroboczek. The babel routing protocol. *RFC 6126*, 2011.

[7] F. R. K. Chung. Diameters of communication networks. *Mathematics of Information Processing*, 1984.

[8] V. Rai S. Ganesh D. Loguinov, A. Kumar. Graph-theoric analysis of structured peer-to-peer systems : Routing distance and fault resilience. *Texas A & M Technical Report*, 2003.

[9] H.Q. Ngo G.W. Peck D.Z. Du, D.F. Hsu. On connectivity of consecutive-d digraphs. *Discrete Mathematics*, 2002.

[10] Steinar H. Gunderson. Global ipv6 statistics. *RIPE 57*, 2008.

[11] Xinyang Zhang Hitesh Ballani, Paul Francis. A study of prefix hijacking and interception in the internet. *ACM SIGCOMM Computer Communication Review*, 2007.

[12] David Karger M. Frans Kaashoek Hari Balakrishnan Ion Stoica, Robert Morris. Chord : a scalable peer-to-peer lookup service for internet applications. *SIGCOMM*, 2001.

[13] J. Trdlicka I. Vrto J. Rolim, P. Tvrdik. Bisecting de bruijn and kautz graphs. *Discrete Applied Math*, 1998.

[14] Luca Saiu Jean-Vincent Loddo. Marionnet, a virtual network laboratory. *FOSDEM*, 2010.

[15] R. Raghavendra K.N. Sivarajan. Fault-tolerant networks based on the bruijn graph. *IEEE Trans. on Computers*, 1991.

[16] Spring N. Lumezanu C., Levin D. Peerwise discovery and negotiation of faster paths. *HotNets*, 2007.

[17] Masaki Itoh Makoto Imase. Design to minimize diameter on building-block network. *Transactions on Computers*, 1981.

[18] David Mazieres Petar Maymounkov. Kademlia : a peer-to-peer information system based on the xor metric.

[19] Andrea Lo Pumo. Scalable mesh network and the address space balancing problem. *University of Cambridge*, 2010.

[20] S.D. Gribble S. Saroiu, P.K. Gummadi. A measurement study of peer-to-peer file sharing systems. *Proc. Multimedia Computing Networking*, 2002.

[21] Hoffman E Snell J Anderson T Savage S, Collins A. The end-to-end effects of internet path selection. *SIGCOMM*, 1999.

[22] S. Toueg W.G. Bridges. On the impossibility of directed moore graphs. *Journal of Combinatorial Theory*, 1980.