ABSTRACT

We present a novel study of reliable broadcast in interdependent networks, in which the failures in one network may cascade to another network. In particular, we focus on the interdependency between the communication network and the power grid network, where the power grid depends on the communication network for control and the communication network depends on the grid for power. In this paper, we propose a best effort broadcast algorithm to handle crash failures in the communication network that may cause cascading failures, where all the correct nodes deliver the message if the sender is correct. At the core of our work is a fully distributed algorithm for the nodes to analyze cascading failures prior to their presence so that failures can be handled accordingly. Our evaluation results show that the algorithm handles cascading failures with little overhead.

1. INTRODUCTION

Modern network services are becoming increasingly dependent on infrastructure networks such that a single failure may cascade to another network and cause the failures of all the dependent networks. Such failures may cascade multiple times in a zigzag manner between the networks and cause widespread failures. A particular example is the interdependency between the power grid and the communication network. The 2003 Italian blackout [15], 2003 U.S. Northeastern power outage [1, 6], and 2011 Southwest blackout [2] are all examples of such interdependency. For instance, during the 2003 U.S. Northeastern power outage, 3,175 communication networks suffered from abnormal connectivity outage [6]. A number of previous efforts focus on the analysis of the robustness of interdependent networks. To the best of our knowledge, none of the previous work formalized the problem between the communication network and other interdependent networks and studied resilient solutions to handle such failures.

We study reliable broadcast in a multihop communication network c-network and a power grid network p-network, which are mutually dependent. The c-network is composed of a set of c-nodes (e.g., routers, sensors, etc.) connected by communication links and the p-network is composed of a set of p-nodes (e.g., power substations) connected by power lines. In order for the nodes to operate, a c-node must receive power from at least one p-node and a p-node must receive control signals from at least one c-node. We model the interdependency using graphs. As illustrated in Figure 1, when c2 fails, it cannot provide control signals to p1 and p3. Node p1 can still operate since c1 has an edge to it. However, p3 fails and it cannot provide power to c3 and c4. Node c3 still operates with power from p2. Since no other p-nodes have edges to c4, c4 fails. In the communication network, a c-node sends a message through certain paths to some c-nodes. C-nodes are subject to crash failures, which can be reliably detected by other c-nodes. In comparison, correct nodes faithfully follow the protocols. Our goal is to design a solution that guarantees best effort broadcast despite the presence of crashing c-nodes, where all the correct nodes deliver the messages if the sender is correct.

We use soft links to handle cascading failures, which are backup links that are activated to handle the failures of primary communication links. The idea of soft links is not new. Specifically, in an independent network, in order to handle the failure of a neighbor, a node ci only needs to maintain a soft link to the neighbor of its neighbor so that messages can still be sent along the path. However, in interdependent net-
works, since failures occur in a widespread cascading fashion, it is possible that the neighbor of c’s neighbor also fails. In fact, it is extremely challenging to determine how many soft links are enough to handle even one single failure without knowing all the cascading effect. Indeed, without a carefully designed algorithm, nodes cannot analyze the cascading failures to handle them. A straightforward solution is to rely on a powerful and centralized computing agent/node to analyze the failures for each node. However, it will cause large communication and computing overhead for the agent since each node must communicate with the centralized agent to learn the result and maintain soft links. Although it might be possible to build a set of distributed agents, it can still cause high communication and maintenance overhead.

We present a best effort broadcast algorithm where nodes analyze cascading failures and maintain the information in a fully distributed manner. At the core of our approach are two key sub-algorithms: f-information collection and link management. The f-information collection is a communication algorithm for the c-nodes to analyze and collect the information of all the cascading failures caused by a single c-node. Based on the f-information of each neighbor, a c-node can learn the next correct c-node in each path so as to maintain soft links. On the other hand, the link management is a mechanism for the nodes to update their routing tables in the presence of failures so that nodes can manage their soft links for long term robustness. As a result, soft links can be correctly maintained and best effort broadcast can be guaranteed if there are no failures during message transmission in the algorithm.

Due to the use of the above approach, best effort broadcast is achieved with the following benefits. First, nodes only need to maintain minimum information in order to analyze the failures. Indeed, since we use a distributed failure analysis algorithm, nodes do not need to maintain the whole network. Second, the information of cascading failures is collected in a fully distributed manner. Last but not least, our algorithm enables very effective usage of soft links. In order to handle one c-node failure, each c-node maintains only one soft link although a set of consecutive c-nodes may fail as a cascading effect. This guarantees that messages can be reliably broadcast to every correct c-node in the communication network. Our evaluation results show that our algorithm achieves low packet drop rate and generates little overhead to the normal network traffic. The tradeoff is a slightly longer delay in handling failures.

Our paper makes the following contributions:

- We present the first reliable broadcast model in the interdependent networks. We study best effort broadcast in the presence of crash failures in the communication network, which may cause cascading failures in both power grid network and communication network.
- We present a fully distributed algorithm for the nodes to analyze the cascading failures. Each node maintains minimum information for the dependent network.
- We use inactive soft links in addition to the primary links in the communication network to achieve best effort broadcast. In order to handle one failure, each c-node only maintains one soft link although multiple cascading failures may occur due to a single failure.
- Our evaluation results show that our algorithm achieves low packet drop rate and generates little overhead to the normal network traffic. The tradeoff is a slightly longer delay in handling failures.

2. RELATED WORK

Modeling interdependencies between critical infrastructure networks is challenging due to a wide range of dimensions such as the type of coupling and type of failures [22, 23]. Previous studies of interdependent network systems focus mainly on the analysis of robustness [3, 12–14, 20, 21, 26]. A number of work study the interdependency between communication network and power grid, most of which focus on finding the vulnerabilities of existing network [21] or the design of a robust topology [13]. In comparison, we study a resilient solution that handles the failures in communication network in the interdependency model.

Reliable broadcast [4] has been widely studied that includes several categories such as regular reliable broadcast and uniform broadcast. We study best effort broadcast, where all the correct nodes deliver the message if the sender is correct. In terms of failures, previous studies tolerate both crash failures [18, 24] and Byzantine (arbitrary) failures [4] in both highly connected network [10, 18] and loosely connected network [7, 19]. We study crash failures in the communication network in the model of interdependent networks.

Reliable broadcast of multipath message forwarding has also been studied in publish/subscribe systems [16, 17]. The
use of soft links has been proposed [17] to handle failures during message forwarding. Each node maintains several soft links that can be activated in the presence of failures. We use similar idea of soft links to handle failures.

Failure detectors were proposed previously to detect faulty behaviors [5, 8]. Chandra and Toueg [5] introduced the notion of unreliable failure detectors, where each failure detector outputs the identity of processes suspected to have crashed and nodes can rely on it for message transmission. We also use failure detectors for c-nodes to detect crashing c-nodes in their routing tables.

3. INTERDEPENDENCY MODEL

We study the interdependency between two networks: the power grid network c-network and the communication network c-network. The power grid consists of a set of n p-nodes p1, p2, · · · , pn (e.g. substations). The communication network consists of a set of m c-nodes c1, c2, · · · , cm (e.g., routers, sensors, etc.). The p-nodes are connected with power lines and the c-nodes are connected with communication links. Each c-node constantly receives power from the p-nodes and every p-node constantly receives control signals from the c-nodes. We follow a model similar to the one used in previous work [9, 13, 21], where a p-node operates if it receives control signals from at least one c-node and a c-node operates if it receives power from at least one p-node. We assume c-nodes do not have backup battery, i.e., a c-node immediately fails if it does not receive power from any p-nodes. In addition, we assume that power substations are connected to power generators, i.e., each power substation is connected to a generator that is sufficient for receiving power and we ignore the amount of power supply or demand. In other words, p-nodes can only fail when there are no incoming control signals.

Table 1: Notations.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vc</td>
<td>all the c-nodes</td>
</tr>
<tr>
<td>Vp</td>
<td>all the p-nodes</td>
</tr>
<tr>
<td>Ec</td>
<td>bidirectional edges between c-nodes</td>
</tr>
<tr>
<td>Ep</td>
<td>bidirectional edges between p-nodes</td>
</tr>
<tr>
<td>Ec,p</td>
<td>directional edges from c-nodes to p-nodes</td>
</tr>
<tr>
<td>Ec,p&amp;Ep</td>
<td>interdependency edges</td>
</tr>
<tr>
<td>oin degree</td>
<td>number of outgoing interdependency edges</td>
</tr>
<tr>
<td>inn degree</td>
<td>number of incoming interdependency edges</td>
</tr>
<tr>
<td>P(Ci)</td>
<td>all the p-neighbors of ci</td>
</tr>
<tr>
<td>C(Pi)</td>
<td>all the c-neighbors of pi</td>
</tr>
<tr>
<td>C(Ci)</td>
<td>all the c-nodes that have interdependency edges to ci</td>
</tr>
<tr>
<td>l1</td>
<td>maximum oin degree of any c-node</td>
</tr>
<tr>
<td>l2</td>
<td>maximum oin degree of any p-node</td>
</tr>
<tr>
<td>l3</td>
<td>maximum inn degree of any p-node</td>
</tr>
<tr>
<td>l4</td>
<td>maximum inn degree of any c-node</td>
</tr>
</tbody>
</table>

Table 1: Notations.

The interdependency between the networks can be represented in a graph G = (V, E), as illustrated in Figure 1. We use several notations to represent the network, as shown in Table 1, and we use edges and links interchangeably.

Vc ∪ Vp is the set of all the nodes and E = Ec ∪ Ep ∪ Ec,p ∪ Ec,p&Ep is the set of all the edges. The network is composed of both directional and bidirectional edges to distinguish different features of both individual network and interdependent networks. Outgoing interdependent network degree (abbreviated as oin degree) and incoming interdependent network degree (abbreviated as inn degree) represent the degree regarding interdependency edges. We also use the term degree by default to refer to the number of edges of a node in an individual network. Without loss of generality, we call two c-nodes neighbors or direct neighbors if there is an edge between them, i.e., they can communicate with each other. The Ec edges are also called primary links. If a c-node ci has an edge to a p-node pi, we call pi a p-neighbor of ci. Similarly, if a p-node pi has an edge to a c-node ci, we call ci a c-neighbor of pi.

Correct nodes faithfully follow the communication algorithm and send messages according to the network protocols. Meanwhile, the c-nodes are subject to crash failures but the crashes can be reliably detected by other nodes. We assume fair-loss links, where if a message is sent infinitely often by a correct sender to a correct recipient, it is received infinitely often. Furthermore, links do not produce spurious messages.

We now introduce several notions and define cascading failures.

Definition 1. (Path) A sequence of c-nodes (c1, · · · , cn) is a path if, ∀i ∈ {1, · · · , n − 1}, ci and ci+1 are neighbors.

Definition 2. (Initial Failure) The failure of a c-node ci is an initial failure if its failure is not caused by the loss of incoming interdependency edges.

Definition 3. (Consecutive Failures) A sequence of c-nodes seq = (c1, · · · , cn) is a set of consecutive failures if, ∀i ∈ {1, · · · , n − 1}, ci fails, n ≥ 2, and seq is a path.

Definition 4. (Single Failure) The failure of a node ci is a single failure if none of its neighbors fails.

Definition 5. (Cascading Failures) A number of nodes s = (c1, · · · , cn, p1, · · · , pn) are cascading failures if, ∃c2, the failure of which makes all the nodes in s lose incoming interdependency edges.

In other words, we refer to the cascading failures as the failures that are caused by the loss of interdependency edges and the nodes that cause the cascading failures as initial failures. For instance, in the example in Figure 1, c2 is the initial failure, c2, c4, and p3 are all single failures, and the set of c3 and p3 are cascading failures caused by c2. In this paper, we seek to handle single crashing initial failures in the c-network, each of which may cause several cascading failures. For other cases, our algorithm can be further extended to handle failures.

We assume each c-node has a perfect failure detector, which provides information about certain c-nodes being crashed or not and it satisfies the following properties.

- Strong Completeness. Eventually, every c-node that crashes is permanently detected by every correct c-node.
- Strong Accuracy. If a c-node c is detected by any c-node, then c has crashed.
The failure detector can be realized using a timeout mechanism. Specifically, in order for a c-node $c_1$ to detect the correctness of c-node $c_2$, it sends a heartbeat message and starts a timer. If $c_1$ has not received a reply message before the timer expires, $c_2$ is suspected to be faulty. Although the failure detector abstraction relaxes the timing assumption on nodes and links [4], performance can be guaranteed under partial synchrony [11]: synchrony holds only after some unknown global stabilization timer, but the bounds on communication and processing delays are themselves unknown to the nodes.

We consider the best effort reliable broadcast problem in c-network under the above interdependency model, where all the correct nodes deliver the messages if the sender is correct. It satisfies the following properties.

- **Validity.** If a correct c-node broadcasts a message $m$ to a set of c-nodes DES, then every correct c-node in DES eventually delivers $m$.
- **No Creation.** If a c-node delivers a message $m$ with sender $s$, then $m$ was previously broadcast by $s$.

### 4. BEST EFFORT BROADCAST

In this section, we first introduce the preliminaries. Then we introduce our best effort broadcast algorithm. Specifically, in addition to primary links, we also use soft links, which are the information of inactive links to handle the failures of primary links. Through the activation of soft links, new connections are built between correct c-nodes so that messages can be reliably delivered in the absence of failures. Notice that if there exists an alternative path to the destination, re-routing might be an option but it can also consume extra communication overhead and cascading failures may occur in the alternative routes.

**Definition 6.** (P-link table PT) For any c-node $c_i$, $PT = \{p_1, \ldots \}$ where $\forall p_j \in PT, p_j \in P(\overline{C_i})$ and $\forall p_k \in P(\overline{C_i})$, $p_k \in PT$.

**Definition 7.** (C-link table CT) For any p-node $p_i$, $CT = \{c_1, \ldots \}$ where $\forall c_j \in CT, c_j \in C(\overline{P_i})$ and $\forall c_k \in C(\overline{P_i})$, $c_k \in CT$.

**Definition 8.** (C-group CG) For any p-node $p_i$, $CG = \{c_1, \ldots \}$ where $\forall c_j \in CG, c_j \in C(\overline{P_i})$ and $\forall c_k \in C(\overline{P_i})$, $c_k \in CG$.

We assume all the c-nodes communicate according to routing tables and refer to it as *regular routing tables*. In other words, for each c-node $c_i$ and a specific destination $c_j$, if $c_i$ wants to send a message to $c_j$, $c_i$ looks up its routing table and verify that $c_j$ is reachable, find a neighbor $c_k$, and sends the message to $c_k$.

In addition to the regular routing tables for message transmission in a regular communication network, each c-node also maintains a p-link table PT, the c-link tables for all the p-nodes in PT, denoted by {CT}, and all the c-groups that it is a member of, denoted by {CG}. For instance, as shown in Figure 2, c-node $c_1$ has a p-link table with $p_1$ in it, the c-link table for $p_1$, and one c-group with $c_1$ and $c_2$. Similarly, c-node $c_2$ has a p-link table with $p_1$ and $p_3$, the c-link tables for both $p_1$ and $p_3$, and a c-group, which only has $c_2$ in it. All the tables can be obtained initially through a heuristic method. We ignore the details in this paper since it is not the main focus of our work.

It is straightforward to see that the number of entries in a p-link table is at most $l_1$ and the number of entries in a c-link table is at most $l_2$, according to the notations in Table 1. Also, a c-node has at most $l_1$ c-groups and the number of c-nodes in each c-group is at most $l_3$. Therefore, in addition to the routing information in a single communication network, the extra storage space for each c-node is $O(l_1 + l_1l_2 + l_1l_3)$, where $l_1$ is the size its p-link table, $l_1l_2$ is the size of c-link tables for the p-nodes, and $l_1l_3$ is the size of all the c-groups.

In the worst case, $l_1$ can be as large as $n$ and $l_2$ and $l_3$ can be as large as $m$. Therefore, the storage space complexity is limited by $O(mn)$.

**Links.** A c-node has two types of links: primary links and soft links. *Primary links* of a c-node are the communication links to the neighbors in the c-network in order to perform message transmission. When the primary links fail, soft links are activated and connections are built with the corresponding c-nodes and messages are transmitted to the activated soft links. Each soft link handles one initial c-node failure where the c-node may cause multiple cascading failures. For simplicity, in this paper, for each c-node, we build one soft link to handle one initial failure of one neighbor for...
each of its path. If there are consecutive initial failures in each path, the algorithm can be further extended to handle the failures.

Failure Detector. As mentioned in §3, each c-node has a built-in failure detector module that provides information about whether certain c-nodes have crashed or not. A c-node uses the failure detector to monitor the correctness of its neighbors (c-nodes), the c-nodes in its c-link tables, and all the c-nodes in its c-groups. Notice that the c-nodes in the c-link tables and c-groups may not be direct neighbors of a c-node. Therefore, the correctness of those c-nodes relies on the neighbors of those c-nodes. Specifically, if a node \( c_i \) wants to learn the correctness of \( c_j \), which is not its direct neighbor, it can learn the correctness of \( c_j \) from the neighbors of \( c_j \). This is because each node must monitor the correctness of its neighbors. However, \( c_i \) does not need to maintain the information of the neighbors of \( c_j \). Instead, it simply sends a \([fd, c_i, c_j]\) message to \( c_j \). When a neighbor of \( c_j \) receives this message, it returns the result to \( c_i \). This applies to all the cases when a c-node needs to monitor a node that is not its direct neighbor. Also notice that if \( c_j \) is not reachable by \( c_i \), it cannot detect the failures.

4.2 Best Effort Broadcast Algorithm

Our best effort broadcast algorithm proceeds as follows. We use a fully distributed f-information collection algorithm (as we will introduce in details in §4.3) for each c-node \( c_i \) to collect the information of cascading failures initialized by \( c_i \). Since the f-information collection algorithm is initialized by each c-node to analyze all the cascading failures caused by itself, the analysis result is then sent to all the neighbors. Based on such information, each c-node \( c_i \) can learn in each path whether the failure of its neighbor will cause other failures in the path. It can then maintain a soft link to the next correct c-node \( c_j \) in each path. The soft link contains the necessary information for \( c_i \) to build a connection with \( c_j \) and the actual connection is made only when the soft link is activated. Notice that in a single communication network, for each soft link of node \( c_i \), it only maintains the information of the neighbor of its neighbor, i.e., the c-node that is two hops away. However, in interdependent networks, if the c-node \( c_i \) that is two hops away from \( c_i \) will also fail, \( c_i \) also needs to learn the identity of \( c_{k} \) the subsequent node of \( c_j \), and analyzes whether \( c_{k} \) will also fail. This process continues until \( c_i \) learns the next correct c-node in the path. For instance, if \( c_i \) learns from its neighbor \( c_j \) that if \( c_i \) and \( c_j \) fails, the subsequent node \( c_{k} \) of \( c_j \) will also fail but the subsequent node \( c_{l} \) of \( c_{k} \) will be correct, \( c_i \) can then build a soft link with \( c_{k} \) in order to handle the failure of \( c_j \). In this case, \( c_i \) also needs to learn from \( c_{k} \) the identity of \( c_i \) since \( c_i \) does not have the information beforehand. After soft links are activated, they become the primary links and the primary links are discarded. New soft links will be maintained after another round of f-information collection.

In the normal case when there are no failures, c-nodes use their regular routing tables for message transmission. If a c-node \( c_i \) detects the failure of its subsequent node \( c_j \) through the failure detector and it has a pending message to \( c_j \), it first diagnoses the situation using the pre-collected f-information from \( c_j \). If the destination node \( c_j \) will also fail if \( c_j \) is faulty, it simply stops broadcasting the message since \( c_i \) will also be faulty. Otherwise, \( c_i \) activates the corresponding soft link, builds the connection, and sends messages to the activated soft link.

For long term robustness, it is important for nodes to monitor the correctness of the c-nodes in the routing tables to maintain the most up-to-date topology. As we will describe in details in §4.4, link management is used to update the tables. The soft links will be updated through another round of f-information collection.

Note that in a mesh topology, it is possible that there are alternative paths to the destination, which we can use to guarantee message delivery. However, it is also possible that failures cascade to the alternative paths. Therefore, the use of soft links is very effective in guaranteeing that all the correct nodes deliver the message as a bottom line. Also note that if some nodes are not initially reachable, the use of soft links is not effective to guarantee best effort broadcast.

```plaintext
1 initialization:
2 \{CT\}, PT ← \{CT\}, PT
3 X.add(c_i) \{output: a set of nodes that will fail\}
4 watchlist()
5 for p_k in PT
6 \( p_k.(iin \ degree) \leftarrow p_k.(iin \ degree) - 1 \)
7 if \( p_k.(iin \ degree) = 0 \) then
8   for \( c_k \) in \{CT\}.p_k
9   \( c_k.(iin \ degree) \leftarrow c_k.(iin \ degree) - 1 \)
10  if \( c_k.(iin \ degree) = 0 \) then
11    watchlist.add(c_k)
12  if \!watchlist().empty()
13    send \{fcollect, c_i, X, \{CT\}, PT\} to watchlist().first
14 on receiving \{fcollect, c_m, X, \{CT\}, PT\} from c_i
15  \{CT\′\} ← merge(\{CT\′\}, \{CT\})
16  PT′ ← merge(PT, PT)
17 X.add(c_i)
18 con ← False
19 prev(c_m) = c_j
20 for p_k in PT′
21 \( p_k.(iin \ degree) \leftarrow p_k.(iin \ degree) - 1 \)
22 if \( p_k.(iin \ degree) = 0 \) then
23   for \( c_k \) in \{CT\′\}.p_k
24   \( c_k.(iin \ degree) \leftarrow c_k.(iin \ degree) - 1 \)
25  if \( c_k.(iin \ degree) = 0 \) then
26    watchlist.add(c_k)
27  con ← True
28 if \!watchlist().empty() then
29   send \{fcollect, c_m, X, \{CT\′\}, PT′\} to watchlist().first
30 if con = False then
31   send \{return, c_m, X, \{CT\′\}, PT′\} to prev(c_m)
32 on receiving \{return, c_m, X, \{CT\′\}, PT′\} from c_j
33  if \( c_j = watchlist().first \) then
34    watchlist().remove(c_j)
35  X ← merge(X, X′)
36 if watchlist().empty() then
37   send \{return, c_m, X, \{CT\′\}, PT′\} to prev(c_m)
38 else
39   send \{fcollect, c_m, X, \{CT\′\}, PT′\} to watchlist().first
```

Figure 3: F-information collection algorithm, where \{CT\}.p_k denotes the c-link table for \( p_k \) in set \{CT\}.

4.3 F-information Collection

F-information collection is for a c-node \( c_m \) to collect the information of the cascading failures by analyzing its fail-
In the case, \( c_{ini} \) is called the initial failure. This \( f \)-information collection algorithm is a distributed message transmission algorithm, as shown in Figure 3. There are two types of messages, \( fcollect \) from \( c_{ini} \) to the nodes that will fail if \( c_{ini} \) fails, and \( freturn \) back to \( c_{ini} \) when no further cascading failures will occur.

The algorithm proceeds as follows. For each initial node \( c_{ini} \), the output is a set of cascading failures \( X \). In the beginning, \( c_i \) (the initial failure) first copies its \( c \)-link tables and \( p \)-link table, as shown in line (ln) 2, and adds itself to \( X \). Next, it looks up its \( p \)-link table and updates the entries by decreasing the \( i \)-in degree by 1, as shown in ln 5-6. If any \( p \)-node \( p_x \) has an incoming interdependency degree of 0, indicating that if \( c_i \) fails then \( p_x \) will also fail, \( c_i \) starts to lookup the \( c \)-link table of \( p_x \), as shown in ln 7-8. Similarly, it also updates the \( c \)-link table by decreasing the \( i \)-in degree by 1, as shown in ln 9. If any \( c \)-node \( c_x \) in the \( c \)-link table of \( p_x \) has \( i \)-in degree of 0, \( c_x \) will also fail if \( c_i \) fails and \( c_x \) is added to the \( watchlist() \). A message with \( c_i \) as the initial node that initializes the \( f \)-information collection, \( X, \{ CT' \} \), and \( PT' \) is sent to the nodes in \( watchlist() \) sequentially, i.e., the message is sent to one node in the \( watchlist() \) at a time, as shown in ln 12-13. Note that the \( p \)-link table and \( c \)-link tables are updated on the copies of the original tables and the goal is to mimic the effect of node failures.

When a node \( c_i \) receives such a message with \( X, \{ CT' \} \), and \( PT' \), it first adds itself to the set \( X \) and copies its \( c \)-link tables and \( p \)-link table. It also merges \( \{ CT' \} \) and \( PT' \) to its tables and updates the common entries, as shown in ln 15-16. The purpose of this step is to let nodes analyze the cascading effect, taking into consideration previous failures. For instance, if \( c_1 \) and \( c_2 \) both have an edge to \( p_1 \) and \( c_1 \) wants to analyze the cascading failures when it fails, it will not include \( p_1 \) in the cascading failures since \( p_1 \) still has an incoming edge when \( c_1 \) fails. However, if in the following \( c_2 \) fails from the failure of another \( p \)-node (but also caused by the initial failure of \( c_1 \)), it must include \( p_1 \) into the cascading failures since now \( p_1 \) has no incoming edges. Therefore, each node must include its copies of \( c \)-link tables and \( p \)-link tables and each \( c \)-node must merge the tables from previous nodes so as to analyze all the failures. The nodes run the same algorithm to analyze the next \( c \)-node failures, as shown in ln 20-29. When a \( c \)-node will not cause any further cascading failures of \( c \)-nodes, it sends a \( freturn \) message with \( X \) to the previous node, as shown in ln 30-31. This process continues until the message reaches the initial node. When the initial node learns the set of cascading failures when it fails, it sends the \( f \)-information to all its neighbors.

In the \( f \)-information collection algorithm, each \( c \)-node only keeps partial information about the cascading failures, i.e., it receives the \( fcollect \) from a \( c \)-node, computes the subsequent failures, and sends to the corresponding \( c \)-nodes that will fail subsequently and waits for \( freturn \) messages. We use a \( watchlist() \) scheme for the nodes to collect the information during such a process. This can be represented as a logical Depth First Search (DFS) tree, as shown in Figure 4, where the arrows between nodes represent the message flow and the links between parent and its child nodes may not be real communication links. Instead, the logical tree just demonstrates the sequence of message transmission during \( f \)-information collection. The root of the tree is the initial node \( c_{ini} \) that starts the \( f \)-information collection process, which is included in both \( fcollect \) and \( freturn \) messages.

When a node \( c_i \) receives a \( fcollect \) message, it keeps the previous node who sent the \( fcollect \) message (the parent in the tree, which may or may not be the initial node), as shown in ln 19) and watches all the \( c \)-nodes that will fail after it fails, i.e., the child nodes in the tree. It sends the \( fcollect \) to one node in its \( watchlist() \) at a time and waits for the \( freturn \) messages. When \( c_i \) receives a \( freturn \) message, it removes the node from \( watchlist() \), as shown in ln 34, and merges \( X' \) to \( X \), as shown in ln 35. If there are still nodes in its \( watchlist() \), \( c_i \) continues to send \( fcollect \) message until it receives \( freturn \) from all of them. When there is no node in \( watchlist() \), it sends a \( freturn \) message to its previous node \( prev(c_{ini}) \). This mechanism is necessary for each node to collect all the cascading failures since each node only carries partial information.

**An Example.** As shown in Figure 4, \( c_1 \) initializes the \( f \)-information collection, where \( X = \{ c_1 \} \) and \( watchlist() = \{ c_2, c_3, c_8 \} \). It first sends a \( fcollect \) to \( c_2 \) and \( c_2 \) further sends a \( fcollect \) message with \( X = \{ c_1, c_2 \} \) to \( c_2 \) and it has \( watchlist() = \{ c_3, c_4 \} \). When \( c_2 \) receives \( freturn \) message from \( c_3 \) with \( X = \{ c_1, c_2, c_3 \} \), it adds \( c_3 \) to its \( X \). Now \( c_2 \) has \( watchlist() = \{ c_4 \} \) and it sends a \( fcollect \) to \( c_4 \). When \( c_2 \) receives \( X = \{ c_1, c_2, c_3, c_4 \} \) from \( c_4 \), it adds \( c_4 \) to its \( X \) and the \( watchlist() \) becomes empty. It can then send a \( freturn \) message to \( c_1 \) and \( c_1 \) has \( watchlist() = \{ c_5, c_6 \} \). Similarly for other branches, each message only contains partial information and each node needs to watch its child nodes one by one until it learns results from all of them. Eventually, \( c_1 \) learns all the results and the \( f \)-information collection is completed.

**The Watchlist.** Notice that we use a sequential mode for nodes to send and collect \( f \)-information, i.e., each node sends \( fcollect \) to one node in its \( watchlist() \) at a time. This is because each message only contains partial information until the information reaches the last node in the tree (\( c_5 \) in the example). Due to the use of the sequential mode, we avoid the case where some failures are not included if the messages are transmitted in parallel to the nodes in the \( watchlist() \). For instance, \( c_4 \) and \( c_6 \) both have an edge to a \( p \)-node \( p_9 \). If the \( fcollect \) messages are sent concurrently to all the child nodes from \( c_4 \), none of \( c_4 \) and \( c_6 \) will consider the failure of \( p_9 \) and therefore some failures may be ignored during this process. In addition, the sequence of sending messages to nodes in the \( watchlist() \) does not affect the result since eventually all the nodes that will fail will be visited.

Additionally, it is possible that some node in the \( watchlist() \) is not reachable. In this case, the \( c \)-node just skips the node and sends the message to next node in the \( watchlist() \). The
reason is that we analyze the failures prior to their presence. For instance, if a node $c_j$ is not reachable at some node $c_i$ during the f-information collection process, node $c_j$ must not be reachable for the initial node $c_2$ due to the fact that $c_2$ is reachable for $c_3$. Therefore, if some nodes are not reachable prior to the failures, the use of soft links is not effective and it is out of scope of this paper.

### 4.4 Link Management

In order for $c$-nodes to maintain the routing tables that represent the most up-to-date topology, each $c$-node $c_i$ monitors the correctness of $c$-nodes in addition to its direct neighbors. These $c$-nodes include the $c$-nodes in the $c$-link tables and $c$-nodes in all the $c$-groups of $c_i$. The algorithm is shown in Figure 5, where $(CG)_{c_j}$ denotes all the $c$-groups that $c_j$ is a member of. $cg.p$ is the $p$-neighbor of the $c$-nodes in $c$-group $cg$. $PT(p)$ is the entry for $p$-node $p$ in the $p$-link table, and $|cg|$ is the number of nodes in $c$-group $cg$. The goal of monitoring the $c$-nodes in the $c$-groups is to update the $p$-link tables since the incoming interdependency degree of the $p$-nodes must be updated. The goal of monitoring the $c$-neighbors of the $p$-neighbors of $c_i$ is to update the $c$-link tables.

```
1 on event $c_j$ is faulty
2 if $c_j$ in $\{CT\}$ then
3 for $c_l$ in $\{CT\}_{c_j}$
4 ct.remove($c_j$)
5 if $c_j$ in $CG$ then
6 for $c_g$ in $(CG)_{c_j}$
7 if $c_i = cg$.leader then
8 send [cgupdate, $c_j$] to $cg$
9 else
10 send [le, $c_j$, $c_k$] to $cg$
11 $PT(cg.p).(iin degree) ← PT(cg.p).(iin degree) - 1$
12 if $|cg| = 1$ then
13 send [$sf, cg.p$] to $(CT).cg.p$
14 if $c_j, p$ in $F$ then
15 $CT(p).iin degree ← CT(p).iin degree - 1$
16 on receiving [cgupdate, $c_j$]
17 $p_i ← (CG).c_j.p$
18 $PT(p_i).iin degree ← PT(p_i).iin degree - 1$
19 on receiving [$sf, c_j, p$]
20 $F$.add($c_j, p$)
```

**Figure 5:** Link management algorithm.

The key idea for the failure detection is that if a $c$-node fails, we must remove the interdependency edges and update the tables at all the applicable $c$-nodes. When the outgoing interdependency edges of a $c$-node are removed, the corresponding number in the $p$-link table must be updated, i.e., the incoming degree of the $p$-node in all the applicable $p$-link tables must be decreased by one. Therefore, we introduce the idea of $c$-groups and we now introduce the maintenance of $c$-groups using a leader-based scheme.

In each $c$-group, a leader is elected and agreed by all the $c$-nodes. Initially, there is a default leader in each group. The leader monitors the correctness of all the $c$-nodes in the same group. When it detects the failure of some $c$-node $c_j$, it updates its $p$-link table and notifies other $c$-nodes, as shown in ln 17-18. Other $c$-nodes then simply update their $p$-link tables, as shown in ln 16-17. If a $c$-node $c_i$ is not the leader in a $c$-group, it monitors the correctness of the leader. If the leader fails, $c_i$ notifies all the nodes in the $c$-group with the id of the new leader $c_k$, as shown in ln 9-10. The leader is elected according to the ids in a deterministic rotating manner. When a node receives or has sent a [le] message, it stores the information of the new leader. If the node is the new leader, it sends a message to all the nodes in the $c$-group and starts monitoring the correctness of them. In addition, all the nodes also update their $p$-link tables since the previous leader has failed.

On the other hand, when $c_i$ fails, we must remove the incoming edges. Therefore, it is straightforward for a $c$-node $c_j$ to monitor the $c$-nodes in its $c$-link tables since the $c$-link tables it maintains are the $c$-link tables of the $p$-nodes in its $p$-link table. In this case, if a $c$-node $c_i$ in the $c$-link table(s) fails, $c_i$ simply removes the corresponding entry, as shown in ln 4 in Figure 5.

If the failure of the $c_i$ will cause the failures of some $p$-nodes but the failure will not cascade to the $c$-network again, we should also update the $c$-link tables for all the applicable $c$-nodes. For this purpose, we add another message type called [$sf$] where if node $c_i$ becomes the only node in a $c$-group $cg$, it sends a [$sf$] message to the nodes in the $c$-link table of node $cg.p$ (the $p$-neighbor of $c$-nodes in $cg$), as shown in ln 12-13. When a node $c_i$ receive a [$sf$] message from some node $c_j$, it starts monitoring the correctness of $c_j$ and also adds $c_j$ to a set $F$, as shown in ln 19-20. If it detects the failure of a node in $F$, it decreases the degree of $c_j$ in the $c$-link table, as shown in ln 14-15.

Assume that a $c$-node can be the leader of at most $t$ $c$-groups, the number of $c$-nodes a $c$-node $c_i$ needs to monitor is limited by $O(l_4 + l_2 + t_3)$, where $l_4$ is the maximum number of neighbors $c_i$ needs to monitor, $l_2l_3$ is the maximum number of $c$-nodes in the $c$-link tables of $c_i$, and $l_3$ is the maximum number of $c$-nodes $c_i$ needs to monitor in its $c$-groups. Since $l_4$ and $t$ can be as large as $n$ and $l_2$ to $l_3$ can be as large as $m$. The number of $c$-nodes a node needs to monitor is limited by $O(mn)$.

### 4.5 Correctness

We show the correctness of our best effort broadcast algorithm in the following theorem and we briefly show the proof. In the theorem, new failures refer to the failures of nodes involved in the f-information collection and link management process. Our approach guarantees correctness if no failures occur in the f-information. The idea is straightforward. F-information collection is used to analyze the cascading failures. Failures during the algorithm will cause inaccurate results and soft links may not be correctly maintained.

**Theorem 1.** Let there be no consecutive initial failures in each path. Best effort reliable broadcast is achieved if there are no new failures among the nodes that participate in each f-information collection and link management process.

**Proof.** The no creation property is straightforward. Since we assume nodes can only fail by crashing, each message, if received by some $c$-node, must be generated by the sender, i.e., no creation property is true.

We now show the validity property in two steps. We first show that if soft links are correctly maintained and there are no consecutive failures, messages are delivered to all the correct receivers. Then we show that if there are no new failures during f-information collection and link management,
soft links can be correctly maintained. Based on these, the validity property can be proved.

We assume that there are no consecutive initial failures. Therefore, one soft link for each node, if correctly maintained, is sufficient to guarantee that messages are reliably delivered to the next correct c-node. For each path, by induction, messages can always be sent along the path to the destination. The destination, if correct, will deliver the message according to the algorithm. Since \( c_n \) is correct, it sends the message to the paths to all the correct destinations. Therefore, the statement is true.

Then we show that if there are no new failures during f-information collection, soft links can be correctly maintained. First, during link management, since we use perfect failure detectors, faulty c-nodes will eventually be detected by correct c-nodes. As discussed in §4.4, we update the p-link tables through the use of c-groups and we update the c-link tables by monitoring of c-nodes in the c-link tables and the use of \([sf]\) messages. If there are no failures during message transmission, all the routing tables will eventually be correctly maintained by all the c-nodes to reflect the most up-to-date topology. Second, since the routing tables are correctly maintained, during f-information collection algorithm, each node is able to analyze the failures. The f-information collection algorithm eventually visits all the c-nodes, if they are connected before the failures, that will fail initialized by each c-node. Therefore, if there are no new failures among all the nodes involved in each f-information collection process, f-information collection enables each c-node to analyze the cascading failures. The correctness of the theorem then follows.

### 4.6 Discussion

The correctness of soft links is guaranteed if there are no new failures during f-information collection and link management. In large-scale and highly dependent and dynamic networks, it may not be the case. In this case, nodes may maintain out-of-date c-link and c-group tables and cause wrong analysis results. It is also possible that the f-information collection algorithm halts where a node waits for a \textit{return} message but some nodes during message transmission fail. As we will discuss further in §5, this does not guarantee best effort broadcast where some messages are not delivered to all the correct destinations. We can handle this problem by also using alternative routes for message delivery to increase the delivery rate and adding timers during f-information to ensure that the algorithm will end in the presence of failures.

As discussed previously, the storage complexity for each node is \( O(l_1 + l_2 + l_3) \) and the complexity for the number of nodes each failure detector module is \( O(l_1 + l_2 + l_3) \). In the worst case, both are limited by \( O(mn) \). It is not hard to conclude that in the worst case of a highly connected mesh topology or highly dependent networks, a c-node might eventually need to maintain the information of all the networks and monitor the correctness of all the c-nodes. However, in this case, it is less possible that a failure of a c-node will cause multiple cascading failures.

### 5. Evaluation

We implement and evaluate our algorithm using OMNeT++ network simulation framework [25]. We construct various sizes of graphs and compare the performance of our distributed algorithm (abbreviated as DA) with the Baseline the Soft Link (abbreviated as SF), both in a single communication network. Baseline is a regular routing algorithm where nodes use routing tables for message transmission and do not use soft links. SL builds soft links between nodes that are two hops away. We limit the number of sink nodes to fewer than three and each node generates a packet by doubling the previous period (i.e., 0.01ms, 0.02ms, 0.04ms, etc.). The average delay between two neighbors is set to 0.01ms.

When failure detectors are used, each node sends a heartbeat message every 0.3ms and the timer is set to 0.1ms. We set up the maximum outgoing interdependency edges of each node to evaluate the interdependent networks with different dependency level. After the interdependent networks are generated, we check the validity of them by ensuring every node has at least one incoming interdependency edge.

We first assess the network traffic according to message types to evaluate the overhead of our algorithm. We observe that the regular network traffic in SL and our proposed algorithm are in general lower than the Baseline. However, the failure detection and f-information do not decrease the regular traffic to a large degree. We then evaluate the robustness of the algorithm by measuring the percentage of packet drop and the average delay of failure detection. We have proved that best effort broadcast can be guaranteed if there are no new failures during f-information collection and link management. However, as discussed in §4.6, in a large-scale and dynamic network where failures are frequent, new failures can occur and best effort broadcast may not be guaranteed. Therefore, we also evaluate the packet drop rate to assess the efficiency of our algorithm. We notice that our proposed algorithm has largely reduced the packet drop rate and there is a tradeoff of a longer failure detection delay. Lastly, we evaluate the f-information collection delay using various sizes of topologies. We find that due to the way we model the networks, the performance is more related to the number of interdependency edges rather than the degree in c-network. Instead, the average degree of c-network determines how many alternative paths exist for message re-routing. We observe that there is no generic relationship between the number of nodes and the average latency for f-information collection. This indicates that f-information collection process does not increase the overall complexity in a scalable network.

#### 5.1 Failure Handling Overhead

We assess the network traffic of different message types and compare our algorithm with Baseline and SL. This is used to assess the overhead caused by our algorithm for distributed failure analysis. In this experiment, we use 200 c-nodes and 200 p-nodes with maximum oin degree of 2. In the c-network, we generate a random mesh graph where the average degree of each node is 3. Each node has 0.1 probability of being crashed. As observed in Figure 6(a), the Baseline algorithm only has regular traffic. Comparably, since SL uses failure detector for each node to monitor the correctness of its neighbors, the regular traffic is in general lower than Baseline. However, the failure detection traffic is relatively stable. This is because each node usually has a fixed number (unless failures occur) of neighbors to monitor.

Compared to Baseline and SL, our proposed DA algorithm generates higher volume of traffic since each node needs to monitor the correctness of a larger number of nodes. Notice that as discussed in §4.6, the c-nodes that a failure detector
needs to monitor may not be the direct neighbors, where the correctness of the c-nodes is monitored by their neighbors. We count these notification messages also as failure detection traffic. As shown in Figure 6(b), the regular traffic is lower than both Baseline and SL and the failure detection traffic is higher than that of SL. The f-information collection traffic is very high in the beginning. This is because all the nodes need to run f-information collection in the beginning of the experiment. After the initialization, f-information collection is run only when failures occur.

![Image](a) Network traffic of different message types for Baseline and SL. (b) Network traffic of different message types for the proposed algorithm. (c) Average tree depth and average number of hops in f-information collection.

![Image](c) Average tree depth and average number of hops in f-information collection.

**Figure 6: Evaluation of the algorithms.**

We also evaluate the packet drop rate and the average failure detection delay with 300 c-nodes and 300 p-nodes. According to Table 2, the packet drop rate for Baseline is similar with previous case and the packet drop rate for SL slightly increases. This is because in SL each node simply monitors the correctness of it neighbors. Although our proposed DA algorithm still achieves the lowest packet drop rate, the packet drop rate is larger than the case with fewer nodes. Also, since the topology has a larger number of nodes, the failure detection delay is also larger, especially with the nature of chain-based topology where there might be a large number of hops between any two nodes. Therefore, it is not hard to conclude that in highly interdependent and large-scale networks, the failure detection delay and packet drop rate can further be increased.

**Figure 7: Average delay for f-information collection.**

In order to evaluate the performance of the algorithms under failures, we employ a chain-based topology for c-network with 50 nodes where the nodes are organized sequentially and each node is connected to at most two other nodes. There are 50 p-nodes and the maximum oin degree is set to 2 for both c-nodes and p-nodes. We set up the sink nodes to be the middle of the chain and only nodes in the first half of the chain may fail. As shown in Table 2, since the Baseline does not have a scheme to handle failures, the packet drop rate is high. This can be explained by the fact that each node becomes critical in message transmission. SL has a much lower packet drop rate because it maintains soft links between nodes that are two hops away, which are still effective when the cascading failures do not include too many consecutive failures. Our proposed DA algorithm achieves the lowest packet drop rate since it handles cascading failures. The tradeoff is a slightly longer failure detection delay. Since each node needs to monitor the correctness of a larger number of nodes, the failure detection generates much longer delay due to the communication overhead.

<table>
<thead>
<tr>
<th>#Nodes</th>
<th>Algorithm</th>
<th>%Packet Drop</th>
<th>Avg FD Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Baseline</td>
<td>52.19%</td>
<td>N.A.</td>
</tr>
<tr>
<td></td>
<td>SL</td>
<td>28.45%</td>
<td>0.19ms</td>
</tr>
<tr>
<td></td>
<td>DA</td>
<td>3.03%</td>
<td>0.55ms</td>
</tr>
</tbody>
</table>

Table 2: Packet drop rate and average failure detection delay of the algorithms

5.2 Robustness

In order to evaluate the performance of the algorithms under failures, we employ a chain-based topology for c-network with 50 nodes where the nodes are organized sequentially and each node is connected to at most two other nodes. There are 50 p-nodes and the maximum oin degree is set to 2 for both c-nodes and p-nodes. We set up the sink nodes to be the middle of the chain and only nodes in the first half of the chain may fail. As shown in Table 2, since the Baseline does not have a scheme to handle failures, the packet drop rate is high. This can be explained by the fact that each node becomes critical in message transmission. SL has a much lower packet drop rate because it maintains soft links between nodes that are two hops away, which are still effective when the cascading failures do not include too many consecutive failures. Our proposed DA algorithm achieves the lowest packet drop rate since it handles cascading failures. The tradeoff is a slightly longer failure detection delay. Since each node needs to monitor the correctness of a larger number of nodes, the failure detection generates much longer delay due to the communication overhead.

![Image](a) Average delay for f-information collection.

5.3 F-information Collection Delay

In order to evaluate the performance of the algorithm, we assess the average delay of f-information collection process using topologies of various network sizes with 5 to 500 c-nodes and p-nodes. We generate random mesh topologies where the average degree of c-nodes is 3. A benchmark x-y represents a graph with x c-nodes, x p-nodes, and the maximum oin degree is y. Based on our observation, when the oin degree is bigger than 4, it is less possible that a failure of a c-node causes multiple failures, i.e., soft links between nodes that are two hops away are sufficient. Also, it is straightforward that if each node only has one outgoing
interdependency edge, the networks become highly interdependent, where a single failure causes the failures of almost the whole network. We also show the depth of the tree and the average number of actual hops during each f-information collection in Figure 6(c), which might not be the same with the number of nodes in the tree since the parent node and a child node may not be direct neighbors. We notice that there is not a generic relationship between the number of nodes and the number of hops due to the way we link the nodes. We also notice that the average depth of the tree is 2 to 3. This indicates that an initial failure of a c-node will only cause failures of some c-nodes and the failures will not cascade furthermore. Additionally, as shown in Figure 7, we also evaluate the average latency for f-information collection. Each f-information collection runs for 0.05 to 0.34ms, where the average latency is directly related to the number of actual hops due to the fact that our algorithm essentially visits all the nodes that will fail. This can be explained by the fact that the f-information collection visits the nodes in a DFS manner so that the delay is directly related to the number of nodes that will fail and the distance between them.

6. CONCLUSION

In this paper, we study best effort broadcast in the interdependent networks between a multihop communication network and a power grid network. We handle crash failures through the use of soft links in the communication network. In order to efficiently build soft links to handle cascading failures, we present a fully distributed algorithm for the nodes to analyze the failures. Each node needs to maintain minimum extra information, which is updated from time to time to reflect the up-to-date network topology. Based on our evaluation results, our algorithm is effective in handling cascading failures with little overhead.

7. REFERENCES


