Best Effort Broadcast under Cascading Failures in Interdependent Critical Infrastructure Networks

Sisi Duan[1]
University of Maryland, Baltimore County
Sangkeun Lee, Supriya Chinthavali, Mallikarjun Shankar[2]
Oak Ridge National Laboratory

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 a communication network and a 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. We guarantee that all the correct nodes, which operate correctly according to the protocol and do not experience any software or hardware or network failures, eventually deliver the message if the sender is correct. We provide a centralized algorithm and a fully distributed algorithm for nodes to analyze and handle cascading failures. At the core of our work is the fully distributed algorithm which enjoys great performance and scalability. Our evaluation results show that the algorithm handles cascading failures with low overhead.

Keywords: Interdependent networks, best effort reliable broadcast, cascading failures, crash failures, soft links

---

[1] {sduan}@umbc.edu.
[2] {lees4, chinthavalis, shankarm}@ornl.gov.

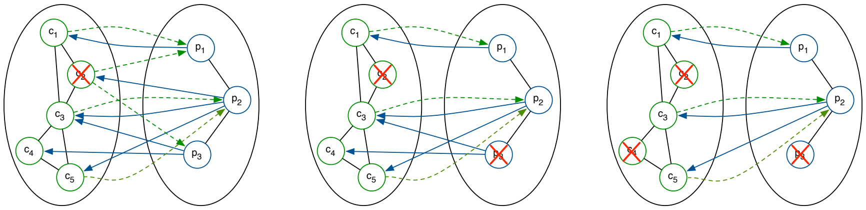
Preprint submitted to Pervasive and Mobile Computing *November 7, 2017*

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 [1], 2003 U.S. Northeastern power outage [2, 3], 2011 Southwest blackout [4], and 2012 Hurricane Sandy [5, 6] 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 [3].

With the development of the Internet of Things (IoT) technologies, communication network plays an increasingly important role in today's infrastructure networks. Indeed, although it has been shown to greatly increase the resilience of existing infrastructures, failures or cyber attacks may cause severe impact to all the infrastructures. However, despite the significance of resilience in interdependent cyber infrastructures, to the best of our knowledge, none of the previous work has formalized the problem of cyber interdependencies 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 c_2 fails, it cannot provide control signals to p_1 and p_3. Node p_1 can still operate since c_1 has an edge to it. However, p_3 fails and it cannot provide power to c_3 and c_4. Node c_3 still operates with power from p_2. Since c_4 does not have any incoming edges, c_4 fails. In the



(a) c_2 fails. It cannot provide control signals to p_1 and p_3.

(b) c_1 has a link to p_1 and p_1 operates. p_3 fails.

(c) p_2 has a link to c_3 and c_3 operates. c_4 fails.

Figure 1: Cascade of a single failure in the interdependent model between a c-network of 5 c-nodes and a p-network of 3 p-nodes.

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, where correct node is a commonly used term in distributed systems and networks [7, 8, 9, 10, 11]. 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. In addition, our work can be easily extended to handle the failures of p-nodes which are not caused directly by any c-node failures.

We illustrate the idea of using 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 c-network, in order to handle the failure of a neighbor, a node c_i only needs to build a soft link to the neighbor of its neighbor prior to the failure so that messages can still be sent along the path when the failure occurs. However, in interdependent networks, since failures occur in a widespread cascading fashion, it is possible that the neighbor of $c_i's$ neighbor also fails. A straightforward solution is to build multiple soft links to different nodes in each path to handle such a problem. However, it is extremely challenging to determine how many soft links are necessary to handle even one single failure without knowing all the cascading effect. Indeed, nodes need to analyze the cascading failures to handle them. One way is to rely on a powerful and centralized computing agent/node to analyze the failures for each node. However, it may incur large communication and computing overhead since each node must communicate with the centralized agent to learn the results. 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 pre-analyze and collect the information of all the cascading failures that can be 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 prior to the actual failures. 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 information of the whole network. Second, the information of cascading failures is collected in a fully distributed manner. Last but not least, our algorithm provides decision makers a reference of highly effective usage of soft links to handle failures prior to their occurrence. 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. In order to fully assess the distributed algorithm, we also propose a centralized algorithm where nodes all rely on a centralized computing agent to analyze cascading failures. Our evaluation results show that our distributed algorithm achieves low packet drop rate and generates little overhead to the normal network traffic. The trade-off 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 of the interdependent networks.
- We illustrate the usage of 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 trade-off 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 types of coupling and the types of failures [12, 13]. Previous studies of interdependent network systems focus mainly on the analysis of vulnerabilities or robustness of the CIs [14, 15, 16, 17, 18]. A few mathematical frameworks [19, 20, 21] and interdependency models [22, 23, 24] have been proposed to support vulnerability analysis. The idea is to mainly predict the catastrophic consequences under extreme events given that live monitoring of the infrastructures could be provided [25, 26, 27, 28, 29, 30]. A number of works study the interdependency between communication network and power grid, most of which focus on finding the vulnerabilities of existing network [22] or the design of a robust topology [23]. In comparison, we study a resilient solution that handles the failures in communication network in the interdependency model. A shorter version of our paper has appeared previously at ICDCN 2017 [31].

Reliable broadcast [7], such as reliable broadcast and uniform broadcast, has been widely studied in independent networks. In terms of failures, previous works study crash failures [8, 9] and Byzantine (arbitrary) failures [7, 10] in both highly connected networks [8, 32, 33, 34] and loosely connected networks [35, 11, 36]. We study best effort broadcast, where all the correct nodes deliver the message if the sender is correct in interdependent networks.

Reliable broadcast of multipath message forwarding has also been studied in publish/subscribe systems [37, 38]. The use of soft links has been studied to handle failures during message forwarding [37]. Each node maintains several soft links prior to the failures 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 in distributed systems and networks [39, 40]. Chandra and Toueg [39] introduced the notion of unreliable failures 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 *p-network* and the communication network *c-network*. The p-network consists of a set of n *p-nodes* $p_1, p_2, \cdots, p_n$ (e.g. substations). The c-network consists of a set of m *c-nodes* $c_1, c_2, \cdots, c_m$ (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 with the one used in previous work [23, 22, 24], 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.

Notation	Meaning
$V_c \& V_p$	all the c-nodes and all the p-nodes, separately
$E_c \& E_p$	bidirectional edges between c-nodes and p-nodes, separately
E_{cp}	directional edges from c-nodes to p-nodes
E_{pc}	directional edges from p-nodes to c-nodes
$E_{cp} \& E_{pc}$	interdependency edges
oin degree	number of outgoing interdependency edges
iin degree	number of incoming interdependency edges
l_1	maximum *oin* degree of any c-node
l_2	maximum *oin* degree of any p-node
l_3	maximum *iin* degree of any p-node
l_4	maximum degree of any c-node

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. $V = V_c \cup V_p$ is the set of all the nodes and $E = E_c \cup E_p \cup E_{cp} \cup E_{pc}$ is the set of all the edges. The network is composed of both directional and bidirectional edges to distinguish different features of independent/single network and interdependent networks. **Outgoing interdependent network degree** (abbreviated as *oin* degree) and **incoming interdependent network degree** (abbreviated as *iin* degree) represent the degree regarding the number of interdependency edges. We also use the term **degree** by default to refer to the number of edges of a node in an independent 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 E_c edges are also called **primary links**. If a c-node c_i has an edge to a p-node p_i, we call p_i a **p-neighbor** of c_i. Similarly, if a p-node p_i has an edge to a c-node c_i, we call c_i a **c-neighbor** of p_i.

We now introduce several notions and define cascading failures.

Definition 1. *(Path) A sequence of c-nodes* $(c_1, \cdots, c_n)$ *is a path if,* $\forall i \in \{1, \cdots, n-1\}$, c_i *and* c_{i+1} *are neighbors.*

Definition 2. *(Root Failure) The failure of a c-node* c_i *is a root failure if its failure is not caused by the loss of incoming interdependency edges.*

Definition 3. *(Consecutive Failures) A sequence of c-nodes* $seq = (c_1, \cdots, c_n)$ *is a set of consecutive failures if,* $\forall i \in \{1, \cdots, n-1\}$, c_i *fails,* $n \geq 2$, *and seq is a path.*

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

Definition 5. *(Cascading Failures) A number of nodes* $s = (c_1, \cdots, c_n, p_1, \cdots, p_n)$ *are cascading failures if,* $\exists c_i$, *the failure of which makes all the nodes in s lose their incoming interdependency edges.*

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 [7], performance can be guaranteed under partial synchrony [41]: synchrony holds only after some unknown global stabilization time, 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.

3.1. A Case Study based on Figure 1.

According to the definitions, 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 root failures. For instance, in the example in Figure 1, c_2 is the root failure, c_2, c_4, and p_3 are all single failures, and the set of c_3 and p_3 are cascading failures caused by c_2. In this paper, we seek to handle single crashing root failures in the c-network, each of which may cause several cascading failures. For other cases such as the scenario where p-node failures can also be root failures, our algorithm can be further extended to handle failures.

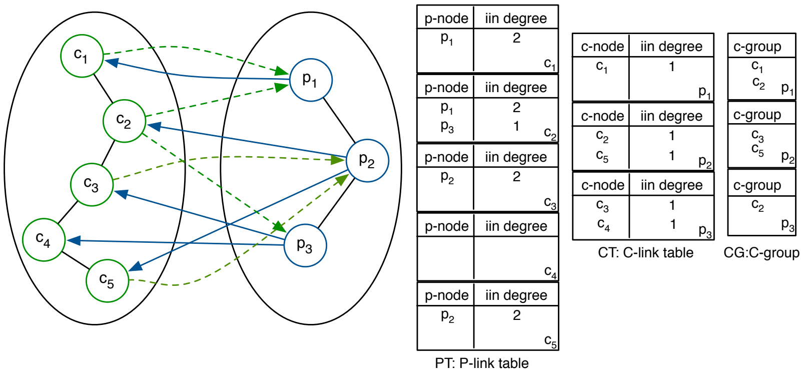
We propose to use **soft links** to handle failures, the details of which will be introduced later in Section 4. Soft links are backup information of links can be maintained or pre-built prior to the actual failures. For example, in order to tolerate the cascading failures caused by root failure c_2, c_1 can maintain a soft link to c_3, which contains the necessary information to build the actual link and does not have to be activated prior to the failures. When c_2 fails and the failure is detected by the failure detector of c_1, the soft link can be activated and messages can be sent towards c_3 if necessary.

We provide a centralized algorithm and a fully distributed algorithm to build soft links to guarantee best effort broadcast in interdependent networks. The design of using soft links can also be considered as a reference of building a robust topology in the interdependent network settings. Such an algorithm can be abstracted away to work in any interdependent networks with similar properties. However, there are several practical problems regarding building soft links. Indeed, in a wireless network, it is feasible to build soft links as backup links and activate them later. In a sensor network, we must also make sure that two nodes are within certain distance so as to make a connection between them. In addition, in a wired network where we consider only routers, we have to actually pre-build the physical links between routers to maintain the soft links. We strengthen that our proposed algorithm can be used as a reference to build a best effort broadcast protocol where if sender is correct and all the soft links can be correctly built or maintained, all the correct receivers will receive the same messages by the sender. Our solution can also be employed with other techniques such as backup routes so as to handle failures.

4. Best Effort Broadcast in Interdependent Networks

In this section, we first introduce the preliminaries and then present our best effort broadcast algorithm. Specifically, in addition to primary links, we also propose to 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 presence of failures.

Our soft link technique is used to guarantee *best effort reliable message transmission* through new communication links so that all the correct destination nodes receive and deliver the same message. In order to correctly build soft links, we employ two sub-algorithms: a cascading failure information collec-



p-node	iin degree
p_1	2

c_1

p-node	iin degree
p_1	2
p_3	1

c_2

p-node	iin degree
p_2	2

c_3

p-node	iin degree

c_4

p-node	iin degree
p_2	2

c_5

PT: P-link table

c-node	iin degree
c_1	1

p_1

c-node	iin degree
c_2	1
c_5	1

p_2

c-node	iin degree
c_3	1
c_4	1

p_3

CT: C-link table

c-group
c_1 c_2

p_1

c-group
c_3 c_5

p_2

c-group
c_2

p_3

CG:C-group

Figure 2: Example of p-link tables, c-link tables, and c-groups.

tion algorithm called *f-information collection* and a *link management* algorithm to update routing tables of the c-nodes. As we will introduce in Section 4.3, we use a fully distributed message transmission algorithm to analyze cascading failures, based on which nodes can maintain soft links. The second part of the distributed algorithm is link management, which is used to update the routing tables when failures are detected, as shown in Section 4.4. We also present a centralized algorithm where a centralized agent analyzes the cascading failures for all the c-nodes. Finally, we discuss and compare the distributed and the centralized algorithms.

4.1. Preliminaries

Routing Tables. As illustrated in Figure 2, each c-node maintains several routing tables, the definitions of which are shown in Definition 6 to Definition 8, where $P(\overrightarrow{c_i})$ represents all the p-neighbors of c_i and $C(\overrightarrow{p_i})$ represents all the c-neighbors of p_i. A *p-link table PT* of a c-node consists of all the p-neighbors and the number of their *iin* degrees. Similarly, for each p-node, there is a *c-link table CT*, which consists of all the c-neighbors and their *iin* degrees. For each p-node p_i, all the c-nodes that have interdependency edges to it form a *c-group CG*.

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

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

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

We assume that all the c-nodes communicate according to the routing tables as in a regular communication network and we refer to the tables 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 verifies that c_j is reachable, finds 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, 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 heuristically during initial network setup. 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 an independent communication network, the extra storage space for each c-node is $O(l_1 + l_1 l_2 + l_1 l_3)$, where l_1 is the size its p-link table, $l_1 l_2$ is the size of c-link tables for the p-nodes, and $l_1 l_3$ 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. The corresponding c-nodes and messages can then be transmitted through the activated soft links. Each soft link is built to handle one root failure (a c-node) where the c-node may cause multiple cascading failures. If there are consecutive root failures in each path, the algorithm can be further extended to handle the failures.

Failure Detectors. As mentioned in Section 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 its 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. 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 message in the format of $[fd, c_i, c_j]$ to c_j, where fd represents the message type (abbreviated for failure detection), c_i represents the sender, and c_j represents the receiver. When a neighbor of c_j receives this message, it returns the result to c_i. Also notice that if c_j is not reachable by c_i, c_i 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 Section 4.3) for each c-node c_i to pre-collect the information of cascading failures caused by the root failure of c_i. Since the f-information collection algorithm is run by each c-node to pre-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

```
**initialization:**
  $\{CT'\}, PT' \leftarrow \{CT\}, PT$          {Works on local copies of tables to analyze failures}
  $X.add(c_i)$                                   {Output: a set of nodes that will fail}
  $watchlist()$                                  {Used to track subsequent cascading failures}
  for $p_x$ in $PT'$                             {Searches p-link table}
    $p_x.(iin\ degree) \leftarrow p_x.(iin\ degree) - 1$
    **if** $p_x.(iin\ degree) = 0$ **then**
      for $c_k$ in $\{CT'\}.p_x$                 {Searches applicable c-link tables}
        $c_k.(iin\ degree) \leftarrow c_k.(iin\ degree) - 1$
        **if** $c_k.(iin\ degree) = 0$ **then**
          $watchlist.add(c_k)$
  if $!watchlist().empty()$                      {There are no subsequent failures}
    send $[fcollect, c_i, X, \{CT'\}, PT']$ to $watchlist().first$
```

Figure 3: F-information collection algorithm (1) – Initialization, where $\{CT\}.p_x$ denotes the c-link table for p_x in set $\{CT\}$.

other failures in the path. Node c_i 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 an independent communication network, for each soft link of node c_i, c_i 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_j 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, c_k will also fail but the subsequent node c_l will be correct, c_i can then build a soft link with c_l 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_l 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 or when faulty nodes are later recovered.

```
on receiving $[fcollect, c_{ini}, X, \{CT'\}, PT']$ from $c_j$
  $\{CT''\} \leftarrow merge(\{CT'\}, \{CT\})$          {Works on copies of c-link table and
  $PT'' \leftarrow merge(PT', PT)$                       merges tables from previous nodes}
  $X.add(c_i)$                                          {Adds itself to $X$ since it will fail if $c_{ini}$ fails}
  $con \leftarrow False$
  $prev(c_{ini}) = c_j$                                 {Tracks previous node}
  for $p_x$ in $PT''$
    $p_x.(iin\ degree) \leftarrow p_x.(iin\ degree) - 1$
    if $p_x.(iin\ degree) = 0$ then
      for $c_k$ in $\{CT''\}.p_x$
        $c_k.(iin\ degree) \leftarrow c_k.(iin\ degree) - 1$
        if $c_k.(iin\ degree) = 0$ then
          $watchlist.add(c_k)$
          $con \leftarrow True$
  if $!watchlist().empty()$ then                       {There are subsequent failures}
    send $[fcollect, c_{ini}, X, \{CT''\}, PT'']$ to $watchlist().first$
  if $con = False$ then                                 {There are no subsequent failures}
    send $[freturn, c_{ini}, X, \{CT''\}, PT'']$ to $prev(c_{ini})$
```

Figure 4: F-information collection algorithm (2) – Handling fcollect message, where $\{CT\}.p_x$ denotes the c-link table for p_x in set $\{CT\}$.

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_l will also fail, c_i simply stops broadcasting the message since c_l 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 Section 4.4, link management is used to update the tables. The soft links will be updated through another round of f-information collection.

Discussion. 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. Also note that if some correct nodes are not initially connected, e.g., under the scenario of network partition, the technique of using soft links is not effective in guaranteeing best effort broadcast.

```
32 on receiving $[freturn, c_{ini}, X', \{CT''\}, PT'']$ from $c_j$
33  if $c_j = watchlist().first$ then      {Receives message from the first node in $watchlist()$}
34     $watchlist().remove(c_j)$
35     $X \leftarrow merge(X, X')$                {Merges results}
36     if $watchlist().empty()$ then     {No further cascading failures for analysis}
37        send $[freturn,c_i,X,\{CT''\},PT'']$ to $prev(c_{ini})$
38     else                              {Continues to collect cascading failures}
39        send $[fcollect,c_i,X,\{CT''\},PT'']$ to $watchlist().first$
```

Figure 5: F-information collection algorithm (3) – Handling $[freturn]$ message, where $\{CT\}.p_x$ denotes the c-link table for p_x in set $\{CT\}$.

4.3. F-information Collection

F-information collection is for a c-node c_{ini} to collect the information of the cascading failures by analyzing its failure. In this case, c_{ini} is called the root failure. This f-information collection algorithm is a distributed message transmission algorithm, as shown in Figure 3, 4, and 5. 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 root node c_{ini}, the output is a set of cascading failures X. In the beginning, c_i (the root 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 iin 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 *iin* degree by 1, as shown in ln 9. If any c-node c_k in the c-link table of p_x has *iin* degree of 0, c_k will also fail if c_i fails. c_k is then added to the *watchlist*(). A message with c_i as the *root 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, c_1 will not include p_1 in X since p_1 still has an incoming edge when c_1 fails. However, if c_2 fails due to the failure of another p-node (but also caused by the root failure of c_1), c_1 must include p_1 in X since now p_2 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 failure, as shown in ln 20-29. When a c-node will not cause any further cascading failures of c-nodes, it sends an $freturn$ message with X to the previous node, as shown in ln 30-31. This process continues until the message reaches the root node. When the root 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 6, 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 root 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 an $fcollect$ message, it keeps track of the previous node who sent the $fcollect$ message (the parent in the tree, which may or may not be the root 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 an $freturn$ 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 $watchlist()$ becomes empty, c_i sends an $freturn$ 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.

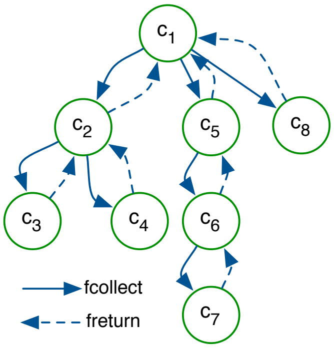


Figure 6: Example of f-information collection.

An Example. As shown in Figure 6, c_1 initializes the f-information collection, where $X = \{c_1\}$ and $watchlist() = \{c_2, c_5, c_8\}$. It first sends an $fcollect$ to c_2 and c_2 further sends an $fcollect$ message with $X = \{c_1, c_2\}$ to c_3 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 an *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 an *freturn* message to c_1 and c_1 has *watchlist*() $= \{c_5, c_8\}$. 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_8 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_1, neither c_4 nor 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 will be visited.

Additionally, it is possible that a 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 occurrence. For instance, if a node c_1 is not reachable by c_2, c_1 must not be reachable by the root node c_3 due to the fact that c_2 is reachable by 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 several 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 7, 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.

```
on event $c_j$ is faulty
  if $c_j$ in $\{CT\}$ then        {Removes c-nodes from c-link tables directly if they are faulty}
    for $ct$ in $\{CT\}.c_j$
      $ct.remove(c_j)$
  if $c_j$ in $\{CG\}$ then        {Updates iin degree for the corresponding p-node of CG}
    for $cg$ in $\{CG\}.c_j$
      if $c_i = cg.leader$ then        {Directly notifies other nodes in the c-groups}
        send $[cgupdate, c_j]$ to $cg$
      else        {Elects a new leader if existing leader fails}
        send $[le, c_j, c_k]$ to $cg$
      $PT(cg.p).(iin\ degree) \leftarrow PT.(cg.p).(iin\ degree) - 1$
      if $|cg| = 1$ then        {$c_i$ becomes the only node in $cg$. }
        send $[sf, c_i, cg.p]$ to $\{CT\}.(cg.p)$  {Notifies all the nodes in c-link table of node $cg.p$}
  if $c_j, p$ in $F$ then
      $CT(p).(iin\ degree) \leftarrow CT(p).(iin\ degree) - 1$
on receiving $[cgupdate, c_j]$
  $p_l \leftarrow \{CG\}.c_j.p$
  $PT(p_l).(iin\ degree) \leftarrow PT(p_l).(iin\ degree) - 1$
on receiving $[sf, c_j, p]$
  $F.add(c_j, p)$
```

Figure 7: 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 *iin* 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 7-8. 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 where the message tag is le (abbreviated for leader election), 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 of c_j are the c-link tables of the p-nodes in $c_j's$ p-link table. In this case, if a c-node c_i in the c-link table(s) fails, c_j simply removes the corresponding entry, as shown in ln 4 in Figure 7.

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 $[sf]$(abbreviated for single failure) where if a 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 $O(l_4 + l_1 l_2 + t l_3)$, where l_4 is the maximum number of neighbors c_i needs to monitor, $l_1 l_2$ is the maximum number of c-nodes in the c-link tables of c_i, and $t l_3$ is the maximum number of c-nodes c_i needs to monitor in its c-groups. Since l_1 and t can be as large as n and l_2 to l_4 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 with 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 analysis results of cascading failures during f-information collection. 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 root 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 root 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_s 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 Section 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 that will fail as a cascading effect, if they were connected before the failures. 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. $\square$

4.6. A Centralized Cascading Failures Analysis Approach

We have presented an f-information algorithm and a link management algorithm for nodes to analyze failures and to maintain the latest topology, which are both fully distributed. An alternative solution for nodes to build soft links is to maintain a centralized computing agent that could analyze cascading failures for the nodes. In this section, we present an algorithm for the centralized agent to analyze the failures.

Each node still maintains its regular routing tables, but not other routing

```
on receiving $[analyze, c_{ini}]$ from $c_{ini}$
  $\{CT'\}, \{PT'\} \leftarrow \{CT\}, \{PT\}$
  $X \leftarrow \emptyset$
  $X' \leftarrow \{c_{ini}\}$
  while $X' \neq \emptyset$
    $c_i \leftarrow X'.first$
    $X'.remove(c_i)$
    for $p_x$ in $\{PT'\}.c_i$
      $p_x.(iin\ degree) \leftarrow p_x.(iin\ degree) - 1$
      if $p_x.(iin\ degree) = 0$ then
        for $c_k$ in $\{CT'\}.p_x$
          $c_k.(iin\ degree) \leftarrow c_k.(iin\ degree) - 1$
          if $c_k.(iin\ degree) = 0$ then
            $X.add(c_k)$
            $X'.add(c_k)$
```

Figure 8: Cascading failures analysis algorithm at a centralized agent C.

tables. The centralized agent maintains regular routing tables, c-link tables, and p-link tables for all the nodes and it does not need to maintain c-groups. This is mainly because c-groups are used for nodes to update their routing tables according to the topology, which are not required for the centralized agent.

Let the centralized computing agent be C. The algorithm works as follows. When a node c_i wants to build soft links, it sends an $[analyze]$ message to C with its identity in it. After C receives the message, it analyzes the cascading failures caused by c_i and sends a message $[cf, \{idx, c\}]$ to c_i (message type cf is abbreviated for cascading failure). In the message, $\{idx, c\}$ represents a map between any index node idx to a c-node c where idx represents a neighbor of c_i and c represents the next correct c-node c towards the direction from c_i to idx. In other words, the identity of c and the corresponding information associated with it represent the soft link c_i needs to build to handle the failure of its neighbor idx.

As shown in Figure 8, the algorithm is similar with the f-information collection algorithm, where for each c-node c_i we simply look up the p-link table and the corresponding c-link tables and find whether there will be cascading failures. The main difference is that f-information collection is usually run by multiple c-nodes to collect all the cascading failures caused by a c-node c_{ini}. In comparison, the centralized agent simply needs to look up all the c-link tables and p-link tables to analyze cascading failures due to the fact that it maintains the topology for all the nodes. We use X to represent all the cascading failures that will be caused by the crash of c_{ini}. X' is used as a buffer for the centralized agent to store the incoming requests.

In order to maintain the most up-to-date topology, the centralized agent also needs to collect information from nodes to update its routing tables. In order to detect the failures of c-nodes, each c-node still needs to monitor the correctness of its neighbors. In the presence of the failure of any of its neighbors, the c-node simply sends a message to the centralized agent. In this paper, we assume that a p-node will fail only when it loses all the control signals from the c-node. Therefore, the centralized agent does not need to collect information of faulty p-nodes. After the centralized agent receives information about a faulty c-node c_i. It simply looks up the p-link table for the c-node, i.e., $\{PT\}.c_i$, and decreases the incoming interdependency degree of c_i by one. If any p-node p_j in $\{PT\}.c_i$ has *iin* degree of zero, c_i further looks up the c-link table for p_j, i.e., $\{CT\}.p_j$, and decreases the *iin* degree by one. This is a similar process with the distributed algorithm, where it continues until all the tables are updated. After a c-node detects a failure, the c-node will also send a request to the agent to obtain information for the new soft links.

4.7. 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, the assumption may not be practical. In the case where new failures may occur when running the algorithms, 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 algorithm halts where a node waits for an $freturn$ message but some nodes during message transmission fail. As we will discuss further in Section 5, this does not guarantee best effort broadcast such that 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 $O(l_1+l_1l_2+l_1l_3)$ and the complexity for the number of nodes each failure detector module needs to detect is $O(l_4+l_1l_2+tl_3)$. In the worst case, both are limited by $O(mn)$. It is not hard to conclude that in the worst case, 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.

Compared to the distributed algorithm, where each node needs to maintain extra information, the centralized algorithm requires the centralized agent to maintain the information of the whole topology and nodes only need to maintain minimum information about its neighbors and monitor the correctness of them. Therefore, the storage space for each node is $O(l_4)$, which is limited by $O(m)$. However, when multiple nodes want to analyze cascading failures simultaneously, high overhead of communication and storage space might be triggered for the centralized agent as the centralized agent has to analyze the cascading failures in each path for all the nodes. Such overhead cannot be avoided during initial network setup phase and will be generated when failures have occurred. Similar with the distributed algorithm, new failures may also occur during the cascading failures analysis. This indicates that the information may also be inaccurate.

Last but not least, our work intends to study cyber impact in interdependent networks. Therefore, we assume that no p-nodes will fail by themselves, i.e., p-nodes will only fail if they lose incoming interdependency edges. However, our algorithm can be easily extended to handle root failures caused by p-nodes.

5. Evaluation

We implement and evaluate our algorithm using OMNeT++ network simulation framework [42]. We construct various sizes of graphs, including both synthetic graphs and graphs constructed from real datasets. We mainly utilize synthetic graphs to test and evaluate the performance of the algorithms under extreme topologies. We also evaluate our algorithm and present a case study based on the graphs constructed from real datasets from HSIP [43].

We compare the performance of our distributed algorithm (abbreviated as DA) with the Baseline and the Soft Link (abbreviated as SF), which are protocols that only work in an independent communication network. Baseline is a regular routing algorithm where nodes use routing tables for message transmission. SL builds soft links between nodes that are two hops away. In addition, we also implement the centralized cascading failures analysis algorithm (abbreviated as CA) and evaluate the overhead of the algorithms and accuracy of routing information.

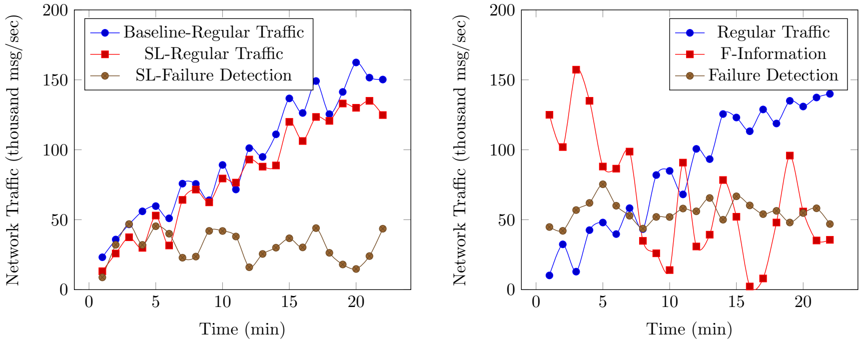
We limit the number of sink nodes to fewer than three and each node generates a packet by doubling the previous period (i.e., 0.01ms, 0.02ms, 0.04ms, etc.). The average delay between two neighbors is set to 0.01ms. When failure detectors are used, each node sends a heartbeat message every 0.3ms and the timer is set to 0.1ms. We set up the maximum outgoing interdependency edges for different graphs to evaluate the interdependent networks with different dependency levels. After the interdependent networks are generated, we check the validity of them by ensuring that every node has at least one incoming interdependency edge.

We first assess the network traffic according to message types to evaluate the overhead of the algorithms. We observe that the regular network traffic in SL, CA, and our proposed DA algorithm are lower than that of the Baseline. However, the failure detection and f-information do not decrease the regular traffic to a large degree. In comparison, although the CA algorithm generates lower traffic for both failure detection and failure analysis, the regular traffic is also much lower than other algorithms. This is mainly due to the fact that all the nodes need to obtain their information through the centralized agent. Since the centralized agent generates high overhead for failure analysis and may not reply to the nodes in a timely manner, the regular traffic is then lowered for all the nodes.

We then evaluate the robustness of our distributed algorithm by measuring the packet drop rate and the average delay of failure detection. Note that we have already 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 Section 4.7, 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 but a longer failure detection delay is incurred. Lastly, we evaluate the f-information collection delay using various topologies. We find that due to the way we model the networks, the performance is highly impacted by the interdependency levels of the two networks. We also 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, SL, and CA. 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 is 3. Each node has 0.1 probability of being crashed. As observed in Figure 9(a), the Baseline algorithm only has regular traffic. In comparison, since SL uses failure detector for each node to monitor the correctness of its neighbors, the regular traffic is lower than that of the Baseline. However, the traffic for failure detection is



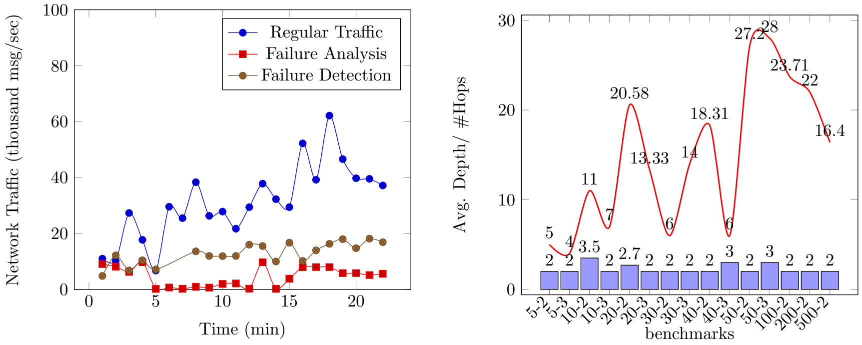
(a) Network traffic of different message types for Baseline and SL.

(b) Network traffic of different message types for the proposed DA algorithm.

Figure 9: Evaluation of the network traffic.

relatively stable. This is because each node has a fixed number of neighbors to monitor unless failures occur. CA relies on a single computing agent to analyze the failures for all the c-nodes. As shown in Figure 10(a), failure analysis and failure detection consistently generate low traffic. The traffic for failure detection is similar with that of SL. This can be explained by the fact that c-nodes in both algorithms only need to monitor the correctness of their neighbors. Also, since nodes all send messages to the centralized agent to obtain information for cascading failures and soft links, it also generates low traffic for failure detection. However, we notice that the regular traffic for CA is much lower than all other three algorithms. This is mainly because when the centralized agent handles concurrent requests from the c-nodes, it has to process the requests sequentially, which causes high overhead and the requests.

Compared to other three algorithms, 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 Section 4.7, 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 traffic for failure detection. As shown in Figure 9(b), the regular traffic is lower than that of both Baseline and SL but the traffic for



(a) Network traffic of different message types for the centralized cascading failures analysis algorithm.

(b) Average tree depth and average number of hops in f-information collection.

Figure 10: Evaluation of the algorithms.

failure detection 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 during network setup in the beginning of the experiment. After the initialization, f-information collection is run only when failures occur.

#Nodes	Algorithm	%Packet Drop	Avg FD Delay
50	Baseline	52.19%	N.A.
	SL	28.45%	0.19ms
	CA	12.35%	0.14ms
	DA	3.03%	0.55ms
300	Baseline	51.80%	N.A.
	SL	32.25%	0.21ms
	CA	14.83%	0.34ms
	DA	13.03%	7.14ms

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 with 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 random nodes close to the middle of the chain. In addition, 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. The CA and our proposed DA algorithm achieve the lowest packet drop rate since they both handle cascading failures. The trade-off for CA is a lower performance as mentioned earlier. The trade-off for DA 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 a much longer delay due to the communication overhead.

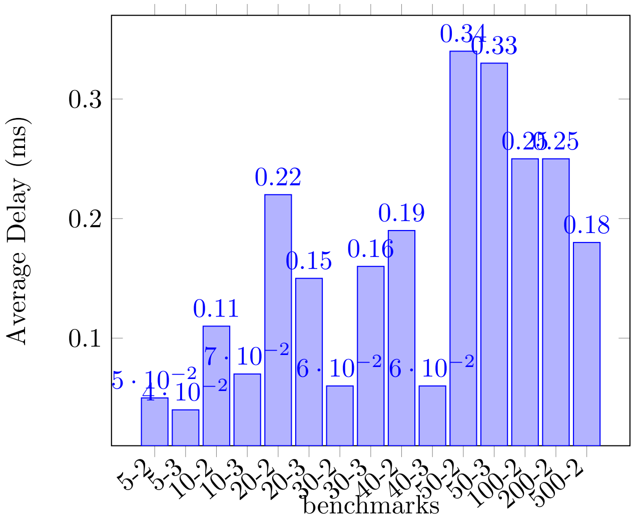


Figure 11: Average delay for f-information collection.

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 longer, 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.

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 the 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 can cause the failures of almost the whole network.

We also show the depth of the tree and the average number of actual hops during f-information collection in Figure 10(b), 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 but it can gets higher in extreme cases. This indicates that a root 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 11, 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.

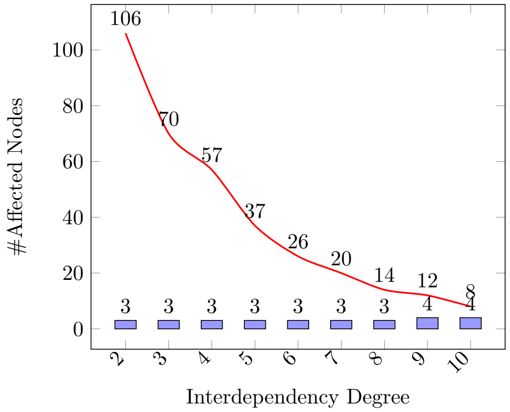


Figure 12: Case Study: Interdependency degree v.s. Affected Nodes. The red line represents the number of c-nodes that will cause cascading failures and the bars represent the average number of affected nodes under each failure.

5.4. A Case Study of Infrastructure Vulnerabilities Based on Geographical Interdependencies

In addition to providing a resilient algorithm that handles cascading failures, our algorithm also serves as a data analysis model to predict and analyze the vulnerabilities of infrastructures. We construct heterogeneous networks based on HSIP datasets [43]. We use cell towers to construct the c-network and electrical substations to construct the p-network. Although our model works specifically well for a smart grid network, it is extremely challenging, if not impossible, to obtain data for the c-network (e.g., control systems infrastructures) as they are usually proprietary. Therefore, we use cell towers for this case study to construct the network and study the interdependencies.

We utilize the geographical locations of the nodes to construct the networks.

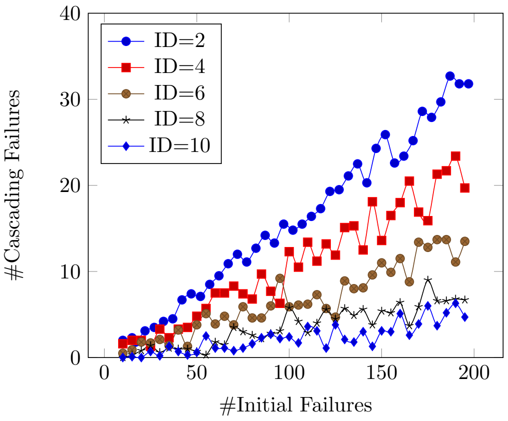


Figure 13: Case study: Number of initial failures v.s. Number of cascading failures. ID represents interdependency degree.

Then we build the interdependency according to the physical distances of the nodes. In order to evaluate the results, we construct graphs with different interdependency degrees, varying from 2 to 10. In other words, for a c-node with x interdependency degree, it has an interdependency edge to x p-nodes that are the closest to it according to geographical locations. Through this process, we are able to analyze the vulnerabilities of the infrastructures based on geographical locations.

We first assess the number of c-nodes, the single failures of which will cause cascading failures for the state of Florida, which has around 900 cell towers and 1000 electrical substations. As observed in Figure 12, the red line represents the number of nodes that can cause cascading failures. In other words, any of those c-nodes is the only node that controls one specific electrical substation. As expected, such a number decreases dramatically as the interdependency degree increases. When the interdependency degree grows to higher than 7, the maximum number of nodes that can cause cascading failures is smaller than 20.

We also evaluate the average number of cascading failures that are caused by single c-node root failures. As observed in Figure 12, the bar chart represents the average number. We found that the average number of cascading failures for each benchmark of interdependency degree is 3 to 4. In other words, single failures will not cause high cascading impact to the infrastructures.

Finally, we also evaluate the number of cascading failures caused by different number of root failures. In this scenario, for each experiment, we set up a number x and select x random root failures and evaluate the number of cascading failures caused by the x root failures. For each x value, we run the experiment 10 times and get the average number of cascading failures. As observed in Figure 13, the number of cascading failures increases as the number of root failures increases. This can be easily explained by the fact that a larger number of nodes will be affected as the number of root failures increases. In addition, as the interdependency degree increases, the number of cascading failures decreases. This is also expected since by increasing the average interdependency degree, the number of cascading failures each node can cause is reduced, i.e., the resilience of the topology is increased.

To summarize, increasing interdependency degree will in general increase the resilience of the networks but it will also increase redundancy in interdependency edges. However, the number of cascading failures for each scenario will not decrease greatly. Furthermore, simply increasing interdependency degree is still not useful to guarantee message delivery during c-node crashes. This indicates that soft links are still necessary in guaranteeing message delivery.

6. Conclusion

In this paper, we study best effort broadcast in the interdependent networks of 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 and a centralized algorithm for the nodes to analyze the failures. Each node needs to maintain minimum information in addition to an independent network, 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.

Acknowledgment

Sangkeun Lee, Supriya Chinthavali, Mallikarjun Shankar were sponsored in part by UT- Battelle, LLC under Contract No. DE-AC05-00OR22725 with the Department of Energy.

References

[1] C. W. Johnson, Analysing the causes of the italian and swiss blackout, 28th september 2003, in: SCS, 2007.

[2] Final report on the august 14, 2003 blackout in the united states and canada: causes and recommendations, Tech. rep., U.S. - Canada Power System Outage Task Force (2004).

[3] J. H. Cowie, A. T. Ogielsk, B. Premore, E. A. Smith, T. Underwood, Impact of the 2003 blackouts on internet communications, Tech. rep., Renesys Corporation (2003).

[4] Arizona-southern california outages on september 8, 2011: causes and recommendations, Tech. rep., Federal Energy Regulatory Commission and the North American Electric Reliability Corporation (2012).

[5] A. Kwasinski, Hurricane sandy effects on communication systems, The University of Texas at Austin, Tech. Rep. PR-AK-0112-2012.

[6] F. E. M. Agency, Hurricane Sandy in New Jersey and New York: Building Performance Observations, Recommendations, and Technical Guidance, Federal Emergency Management Agency, 2013.

[7] C. Cachin, R. Guerraoui, L. Rodrigues, Introduction to reliable and secure distributed programming, Springer, 2011.

[8] L. Lamport, The part-time parliament, ACM Transactions on Computer Systems 16 (2) (1998) 133–169.

[9] R. van Renesse, F. B. Schneider, Chain replication for supporting high throughput and availability, in: OSDI, 2004, pp. 91–104.

[10] M. Castro, B. Liskov, Practical byzantine fault tolerance and proactive recovery, ACM Transactions on Computer Systems (TOCS) 20 (4) (2002) 398–461.

[11] D. Dolev, The byzantine generals strike again, Journal of Algorithms 3 (1) (1982) 14–30.

[12] P. Pederson, D. Dudenhoeffer, S. Hartley, M. Permann, Critical infrastructure interdependency modeling: a survey of us and international research, Idaho National Laboratory (2006) 1–20.

[13] S. M. Rinaldi, J. P. Peerenboom, T. K. Kelly, Identifying, understanding, and analyzing critical infrastructure interdependencies, Control Systems, IEEE 21 (6) (2001) 11–25.

[14] S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, S. Havlin, Catastrophic cascade of failures in interdependent networks 464 (2010) 1025–1028.

[15] J. Gao, S. V. Buldyrev, H. E. Stanley, S. Havlin, Networks formed from interdependent networks, Nature Physics 8 (2012) 40–48.

[16] M. Ouyang, Review on modeling and simulation of interdependent critical infrastructure systems, Reliability Engineering and System Safety 121 (2014) 43–60.

[17] A. J. Holmgren, Using graph models to analyze the vulnerability of electric power networks, Risk Analysis 26 (4) (2006) 955–969.

[18] J. Winkler, L. Dueñas-Osorio, R. Stein, D. Subramanian, Interface network models for complex urban infrastructure systems, Journal of Infrastructure Systems 17 (4) (2011) 138–150.

[19] X. Huang, S. Shao, H. Wang, S. V. Buldyrev, H. E. Stanley, S. Havlin, The robustness of interdependent clustered networks, EPFA 101 (1).

[20] S. V. Buldyrev, N. W. Shere, G. A. Cwilich, Interdependent networks with identical degree of mutually dependent nodes, Physical Review 83 (1).

[21] W. Li, A. Bashan, S. V. Buldyrev, H. E. Stanley, S. Havlin, Cascading failures in interdependent lattice networks: the critical role of the length of dependency links, Physical Review Letters 108 (22).

[22] M. Parandehgheibi, E. Modiano, Robustness of interdependent networks: the case of communication networks and the power grid, in: GLOBECOM, 2013, pp. 2164–2169.

[23] M. F. Habib, M. Tornatore, B. Mukherjee, Cascading-failure-resilient interconnection for interdependent power grid-optical networks, in: OFC, 2015.

[24] S. Duan, S. Lee, S. Chinthavali, M. Shankar, Reliable communication models in interdependent critical infrastructure networks, in: Resilience Week (RWS), 2016, IEEE, 2016, pp. 152–157.

[25] D. Josephsen, Building a monitoring infrastructure with Nagios, Prentice Hall PTR, 2007.

[26] I. Jawhar, N. Mohamed, K. Shuaib, A framework for pipeline infrastructure monitoring using wireless sensor networks, in: WTS 2007, IEEE, 2007, pp. 1–7.

[27] A. M. Barker, E. B. Freer, O. A. Omitaomu, S. J. Fernandez, S. Chinthavali, J. B. Kodysh, Automating natural disaster impact analysis: An open resource to visually estimate a hurricane's impact on the electric grid, in: Southeastcon, 2013, pp. 1–3.

[28] H. M. Supriya Chinthavali, Sankeun Lee, S. Mallikarjun, Real-time social media data analytics for situational awareness of the electric grid.

[29] M. Shankar, J. Stovall, A. Sorokine, B. Bhaduri, T. King, Visualizing energy resources dynamically on earth, in: Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE, IEEE, 2008, pp. 1–4.

[30] S. Lee, L. Chen, S. Duan, S. Chinthavali, M. Shankar, B. A. Prakash, URBAN-NET: A network-based infrastructure monitoring and analysis system for emergency management and public safety, in: Big Data 2016, IEEE, 2016, pp. 2600–2609.

[31] S. Duan, S. Lee, S. Chinthavali, M. Shankar, Best effort broadcast under cascading failures in interdependent networks, in: Proceedings of the 18th International Conference on Distributed Computing and Networking, no. 27, ACM, 2017.

[32] S. Duan, H. Meling, S. Peisert, H. Zhang, BChain: Byzantine replication with high throughput and embedded reconfiguration, in: OPODIS, 2014, pp. 91–106.

[33] S. Duan, S. Peisert, K. N. Levitt, hBFT: Speculative byzantine fault tolerance with minimum cost, IEEE Transactions on Dependable and Secure Computing 12 (1) (2015) 58–70.

[34] S. Duan, K. N. Levitt, H. Meling, S. Peisert, H. Zhang, ByzID: Byzantine fault tolerance from intrusion detection, in: SRDS, 2014, pp. 253–264.

[35] A. Maurer, S. Tixeuil, On byzantine broadcast in loosely connected networks, in: DISC, 2012, pp. 253–266.

[36] S. Duan, L. Nicely, H. Zhang, Byzantine reliable broadcast in sparse networks, in: Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on, IEEE, 2016, pp. 175–182.

[37] R. S. Kazemzadeh, H.-A. Jacobsen, Opportunistic multipath forwarding in content-based publish/subscribe overlays, in: Middleware, 2012, pp. 249–270.

[38] R. S. Kazemzadeh, H.-A. Jacobsen, Reliable and highly available distributed publish/subscribe service, in: SRDS, 2009, pp. 41–50.

[39] T. D. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, Journal of ACM 43 (2) (1996) 225–267.

[40] A. Doudou, B. Garbinato, R. Guerraoui, A. Schiper, Muteness failure detectors: specification and implementation, in: EDCC, 1999, pp. 71–87.

[41] C. Dwork, N. Lynch, L. Stockmeyer, Consensus in the presence of partial synchrony, Journal of ACM 32 (2) (1988) 288–323.

[42] A. Varga, OMNeT++, in: Modeling and Tools for Network Simulation, 2010.

[43] Homeland security infrastructure program (hsip).
URL `https://gii.dhs.gov/HIFLD/hsip-guest`