Fault-Tolerant Consensus in Directed Graphs *

Lewis Tseng
Department of Computer Science
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
ltseng3@illinois.edu

Nitin H. Vaidya
Department of Electrical and Computer Engineering
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
nhv@illinois.edu

ABSTRACT

Consider a point-to-point network in which nodes are connected by *directed* links. This paper proves *tight* necessary and sufficient conditions on the underlying communication graphs for solving the following fault-tolerant consensus problems:

- Exact crash-tolerant consensus in *synchronous* systems,
- Approximate crash-tolerant consensus in *asynchronous* systems, and
- Exact Byzantine consensus in *synchronous* systems.

The problem of asynchronous Byzantine consensus in directed graphs remains open.

Prior work has developed analogous necessary and sufficient conditions for *undirected* graphs [11, 3, 9]. However, the conditions for undirected graphs are not adequate to completely characterize these consensus problems in *directed* graphs. Moreover, the algorithms for *directed* graphs presented in this paper are substantially different from those previously developed for *undirected* graphs. Other prior work on directed graphs has explored somewhat different problems than those solved in this paper.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: [Distributed applications]

General Terms

Algorithm, Theory

*This research is supported in part by National Science Foundation awards 1329681, National Science Foundation awards 1421918 and Army Research Office grant W-911-NF-0710287. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. government.

PODC'15, July 21–23, 2015, Donostia-San Sebastián, Spain.

Keywords

Consensus, directed networks, crash and Byzantine faults, asynchronous and synchronous systems

1. INTRODUCTION

Since Lamport, Shostak, and Pease posed the Byzantine *fault-tolerance* problem [19], it has received significant attention over the past three decades [3, 17]. The fault-tolerant consensus problem has been studied extensively in complete networks (e.g., [19, 10, 1]) and in undirected networks (e.g., [11, 9]). This paper addresses the fault-tolerant consensus problem in *incomplete directed* networks, i.e., not every pair of nodes is connected by a communication channel, and the communication channels are not necessarily bi-directional. Our work is motivated by the presence of directed links in wireless networks. However, we believe that the results here are of independent interest as well. There is also work studying directed graphs for similar problems (e.g., [8, 7, 18]), but the formulation is different from our work as elaborated later in Section 2.

The fault-tolerant consensus problem [3, 17] considers n nodes, of which at most f nodes may be faulty. In our work, we address crash faults and Byzantine faults both. Each node is given an *input*, and after a finite amount of time, each fault-free node should produce an *output*, which satisfies appropriate *validity* and *agreement* conditions. We limit our consideration to *scalar* input and output. For synchronous systems, we consider *exact* consensus, where the *agreement* condition requires the fault-free nodes to agree on exactly the *same* output. Since exact consensus is impossible in asynchronous systems [12], we consider *approximate* consensus, where the *agreement* condition requires the fault-free nodes to produce outputs within a certain constant ϵ ($\epsilon > 0$) of each other.

The goal of this paper is to characterize the necessary and sufficient conditions on the underlying communication graph for solving exact and approximate consensus, respectively, in synchronous and asynchronous systems in the presence of *crash* and *Byzantine* faults. Our problem formulation allows nodes to have complete knowledge of the communication graph. Table 1 identifies the results presented in this paper. Where prior work has already obtained the necessary and sufficient conditions, those conditions are also enumerated. Recall that n is the number of nodes in the system. The table on the left is for consensus with up to f crash faults, whereas the table on the right is for consensus with up to f Byzantine faults. The term *connectivity* is used here to

Table 1: Summary of Tight Conditions

Crash-Tolerant Consensus		
	Synchronous	Asynchronous
Undirected graph	$f+1$- connectivity, $n > f$ (follows from well-known results [17, 3])	$f+1$- connectivity, $n > 2f$ (follows from well-known results [17, 3])
Directed graph	*This work*	*This work*

Byzantine Consensus		
	Synchronous	Asynchronous
Undirected graph	$2f+1$- connectivity, $n > 3f$ [11, 9]	$2f+1$- connectivity, $n > 3f$ (follows from [1, 11])
Directed graph	*This work*	Open problem

mean *node connectivity*. We will often use the terms *graph* and *network* interchangeably.

The problem of asynchronous Byzantine consensus in directed graphs remains open. To prove that the necessary conditions presented in the paper are also sufficient, we have developed algorithms that achieve consensus in graphs satisfying those conditions. It turns out that the algorithms required for *directed* graphs are substantially different from those previously developed for *undirected* graphs [11, 3, 9].

As noted above, our problem formulation allows nodes to have complete knowledge of the network topology. It is also possible to achieve consensus when nodes are *constrained* to only have knowledge of the *local* network topology, particularly their immediate neighbors – such algorithms are beyond the scope of this paper. We note, however, that the algorithms using only the *local* neighborhood information require richer communication graphs. Such algorithms have been a topic of prior work [8, 5, 13, 15], including our own work [23, 24, 20].

2. RELATED WORK

Lamport, Shostak, and Pease first addressed the Byzantine fault-tolerance problem in [19]. Subsequent work [11, 9] characterized the necessary and sufficient conditions under which Byzantine consensus is solvable in *undirected* graphs. However, these conditions are not adequate to fully characterize the *directed* graphs in which Byzantine or crash-tolerant consensus is feasible. For synchronous systems, [8] solves *approximate* crash-tolerant consensus in (dynamic) directed networks using local averaging algorithms – on the other hand, we solve *exact* crash-tolerant consensus in directed networks, which requires a different type of algorithm. In the asynchronous setting, [8] addresses approximate consensus with crash faults in *complete* graphs (which are necessarily undirected) – on the other hand, we solve the problem in *incomplete directed* graphs. We also address Byzantine consensus in synchronous systems.

Previous work also studies graph properties for other similar problems. Bansal et al. [4] identified tight conditions for achieving Byzantine consensus in *undirected* graphs using *authentication*. Bansal et al. discovered that all-pair reliable communication is not necessary to achieve consensus when using authentication. Our work differs from Bansal et al. in that our results apply in the absence of authentication or any other security primitives; also our results apply to *directed* graphs. We show that even in the absence of authentication, all-pair reliable communication is not necessary for Byzantine consensus. Alchieri et al. [2] explored the problem of achieving exact consensus in *unknown* networks with Byzantine nodes, but the underlying communication graph is assumed to be *fully-connected*. In our work, the network is assumed to be known to all nodes, and we consider incomplete directed graphs.

Many researchers in the decentralized control area, including Bertsekas and Tsitsiklis [5] and Jadbabaei, Lin and Morse [13], have explored approximate consensus in the absence of faults, using only near-neighbor communication in systems wherein the communication graph may be partially connected and time-varying. Our work considers the case when nodes may suffer crash or Byzantine failures.

Our prior work [23, 24] has considered a restricted class of iterative algorithms for achieving *approximate* Byzantine consensus in directed graphs, where fault-free nodes must agree on values that are approximately equal to each other using iterative algorithms with limited memory (the state carried by the nodes across iterations must be in the convex hull of inputs of the fault-free nodes, which precludes mechanisms such as multi-hop forwarding of messages). The conditions developed in such prior work are *not* necessary when no such restrictions are imposed. Independently, LeBlanc et al. [15, 16], and Zhang and Sundaram [25, 26] have developed results for iterative algorithms for approximate consensus under a *weaker* fault model, where a faulty node must send *identical* messages to all the neighbors.

Recently, researchers have explored consensus problem in directed dynamic networks [6, 7]. The typical fault model in dynamic networks is named *message adversary*, which controls the communication pattern, i.e., the adversary has the power to specify the sets of communication graphs. Biely et al. studied the exact consensus problem [6] and k-set consensus problem [7] (i.e., at most k different outputs at fault-free nodes) in dynamic networks under *message adversary*, and the system is assumed to be synchronous [7]. All the nodes are assumed to be fault-free in [6, 7].

3. MAIN RESULTS

Before presenting our results, we introduce the system model and some terminology to facilitate the discussion.

System Model.

We consider a point-to-point network in which nodes are connected by *directed* links. The communication network is *static*, and it is represented by a simple directed graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n nodes, and $\mathcal{E}$ is the set of directed edges between the nodes in $\mathcal{V}$. The communication links are reliable. We assume that $n \geq 2$, since the consensus problem for $n = 1$ is trivial. Node i can transmit messages to another node j if directed edge (i, j) is in $\mathcal{E}$. Each node can send messages to itself as well; however, for convenience,

we exclude self-loops from set $\mathcal{E}$. We will often use the terms *edge* and *link* interchangeably.

In the system, up to f nodes may become faulty. A node that suffers crash failure simply stops taking steps. On the other hand, a Byzantine faulty node may misbehave arbitrarily. Possible misbehavior includes sending incorrect and mismatching (or inconsistent) messages to different neighbors. The Byzantine nodes may potentially collaborate with each other. Moreover, the Byzantine nodes are assumed to have a complete knowledge of the execution of the algorithm, including the states of all the nodes, contents of messages the other nodes send to each other, the algorithm specification, and the network topology.

Terminology.

Upper case letters are used to name sets. Lower case italic letters are used to name nodes. All paths used in our discussion are directed paths. Node j is said to be an incoming neighbor of node i if $(j,i) \in \mathcal{E}$. Let N_i^- be the set of incoming neighbors of node i, i.e., $N_i^- = \{j \mid (j,i) \in \mathcal{E}\}$. Define N_i^+ as the set of outgoing neighbors of node i, i.e., $N_i^+ = \{j \mid (i,j) \in \mathcal{E}\}$. For set $B \subseteq \mathcal{V}$, node i is said to be an incoming neighbor of set B if $i \notin B$, and there exists $j \in B$ such that $(i,j) \in \mathcal{E}$. Given subsets of nodes A and B, set B is said to have k incoming neighbors in set A if A contains k distinct incoming neighbors of B.

DEFINITION 1. *Given disjoint non-empty subsets of nodes A and B, $A \overset{x}{\Rightarrow} B$ if B has at least x distinct incoming neighbors in A. When it is not true that $A \overset{x}{\Rightarrow} B$, we will denote that fact by $A \overset{x}{\not\Rightarrow} B$.*

Consider the network in Figure 1, which contains two cliques K_1 and K_2, each consisting of 7 nodes. Within each clique, each node has a directed link to the other 6 nodes in that clique – these links within each clique are not shown in the figure. There are 8 directed links with one endpoint in clique K_1 and the other endpoint in clique K_2. In the network, K_2 has 4 incoming neighbors in K_1, namely u_1, u_2, u_3 and u_4. Thus, $K_1 \overset{4}{\Rightarrow} K_2$. Similarly, $K_2 \overset{4}{\Rightarrow} K_1$.

Main Results.

The main contribution of this work is to identify necessary and sufficient conditions on the underlying communication graphs $G(\mathcal{V}, \mathcal{E})$ for achieving *consensus* in directed networks. We summarize the main results in three theorems below. Each theorem below requires the graph to satisfy a certain condition: we name the conditions in Theorems 1, 2 and 3 as **CCS** (abbreviating Crash-Consensus-Synchronous), **CCA** (Crash-Consensus-Asynchronous) and **BCS** (Byzantine-Consensus-Synchronous), respectively. Characterization of the necessary and sufficient condition for approximate Byzantine consensus in asynchronous systems remains open.

The precise *validity* conditions that need to be satisfied for different versions of the consensus problem are specified at the start of Sections 4, 5 and 6, respectively. In brief, for Theorem 1, the output at the nodes must equal the input of one of the nodes. For Theorem 2, the output must be in the range of the inputs of all the nodes. For Theorem 3, the validity condition depends on whether the inputs are binary or not: for binary inputs, the output must be the input of a fault-free node, whereas for multi-valued inputs, the output must equal the input of fault-free nodes when they all have the same input. These conditions are standard in the related literature [17, 3, 14].

THEOREM 1. *Exact crash-tolerant consensus in a synchronous system is feasible iff for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \leq f$, either $L \cup C \overset{1}{\Rightarrow} R$ or $R \cup C \overset{1}{\Rightarrow} L$.* (Condition CCS)

THEOREM 2. *Approximate crash-tolerant consensus in an asynchronous system is feasible iff for any partition L, C, R of $\mathcal{V}$, where L and R are both non-empty, either $L \cup C \overset{f+1}{\Rightarrow} R$ or $R \cup C \overset{f+1}{\Rightarrow} L$.* (Condition CCA)

THEOREM 3. *Exact Byzantine consensus in a synchronous system is feasible iff for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \leq f$, either $L \cup C \overset{f+1}{\Rightarrow} R$ or $R \cup C \overset{f+1}{\Rightarrow} L$.* (Condition BCS)

The network shown in Figure 1 satisfies Condition BCS for $f = 2$, whereas the network in Figure 2 satisfies Condition CCS for $f = 1$.

Intuitively, for consensus to be achieved, there must be a way for information to "flow between" different subsets of fault-free nodes (subsets L and R in the theorems above), despite the presence of faulty nodes. The different conditions above capture this intuition. Observe that, in each case, for different values of x, we obtain the requirement of the form "either $L \cup C \overset{x}{\Rightarrow} R$ or $R \cup C \overset{x}{\Rightarrow} L$". Intuitively, information must "flow" either from $L \cup C$ to R, or from $R \cup C$ to L, but it is not necessary that the information flows in both directions – this "asymmetry" in the necessary and sufficient condition is a consequence of the directed nature of the communication network. Note that in Condition CCA (in Theorem 2), the partition does not need set F, unlike Conditions CCS and BCS for the synchronous case.

4. EXACT CRASH-TOLERANT SYNCHRONOUS CONSENSUS

An exact crash-tolerant consensus algorithm must satisfy the following three properties: (i) *Agreement*: the output (i.e., decision) at all the fault-free nodes is identical. (ii) *Validity*: the output of each fault-free node equals the input of one of the nodes. (iii) *Termination*: every fault-free node decides on an output in finite amount of time.

Theorem 1 in Section 3 presents the necessary and sufficient condition (named Condition CCS) for solving the above problem in directed graphs.

4.1 Necessity of Condition CCS

LEMMA 1. *Condition CCS is necessary for exact crash-tolerant consensus in a synchronous system.*

PROOF. The proof is by contradiction. Suppose that there exists a consensus algorithm for graph $G(\mathcal{V}, \mathcal{E})$, but $G(\mathcal{V}, \mathcal{E})$ does not satisfy the condition in the lemma. Thus, there exists a node partition F, L, C, R of $\mathcal{V}$ such that (i) $|F| \leq f$, (ii) L and R are both non-empty, and (iii) $L \cup C \overset{1}{\not\Rightarrow} R$ and $R \cup C \overset{1}{\not\Rightarrow} L$. The last two assumptions imply that nodes in $L \cup C$ have no links to nodes in R, and nodes in $R \cup C$ have no links to nodes in L.

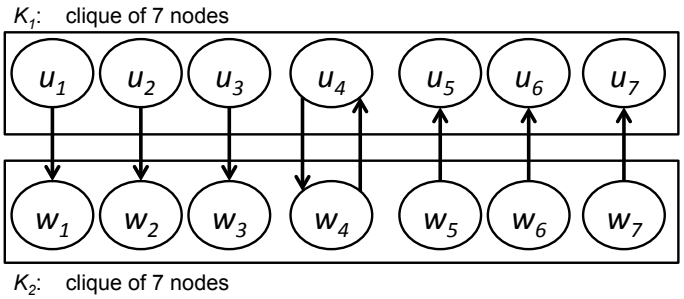

Figure 1: Edges within cliques K_1 and K_2 are not shown.

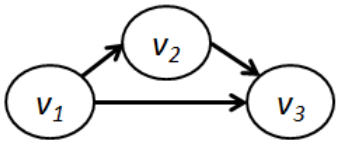

Figure 2: Another example.

Now, consider an execution of the consensus algorithm where F is the set of faulty nodes which crash before the start of the algorithm. All the other nodes are fault-free. This is possible, since by assumption, $|F| \leq f$. Also, suppose that all the nodes in L have input 0, and all the nodes in R have input 1. Nodes in C may have input either 0 or 1.

Consider any node $x \in L$. Since nodes in F fail before taking any steps in the algorithm, and as noted above, there are no links from $C \cup R$ to any node in L, the only input value learned by x throughout the execution of the algorithm is 0. Then to satisfy the validity property, 0 must be the output of node x. Similarly, any node y in R can only learn input values 1 throughout the execution of the algorithm, and thus, 1 must be the output of node y. Since L and R are non-empty and consist of fault-free nodes, the above observations imply that the agreement property is violated. This is a contradiction. □

4.2 Sufficiency of Condition CCS

We prove the sufficiency of Condition CCS constructively by presenting an algorithm, called MVC, and proving its correctness. Algorithm MVC can achieve consensus with multi-valued inputs. MVC uses binary consensus algorithm Min-Max presented below as a component. Algorithm Min-Max can achieve consensus with *binary* inputs (0 or 1). The proposed algorithms prove sufficiency of Condition CCS, but they are not necessarily the most efficient. Development of optimal algorithms needs further research.

4.2.1 Binary Consensus Algorithm Min-Max

Note that Algorithm *Min-Max* has input parameter x_i. To achieve binary consensus, each node i performs Algorithm *Min-Max* passing its binary input value as parameter x_i to Algorithm *Min-Max*. Algorithm Min-Max uses *Compute* as a sub-routine. *Compute* has two parameters: t, which is a binary value, and *Function*, which may be specified as *Min* and *Max*. In the last step of each round in *Compute* at node i, the *Function* is applied to set S_i. $Min(S_i)$ returns minimum of the values in set S_i, and $Max(S_i)$ returns the maximum of the values in set S_i.

Algorithm Min-Max(x_i) for node $i \in \mathcal{V}$

Initialization: $v_i[0]$:= parameter x_i passed to Min-Max

- For phase number $p := 1$ to $2f + 2$:

  If p *mod* $2 = 0$, then (**Min Phase**)

  $v_i[p]$:= Compute $(v_i[p-1],\ Min)$

  Else, (**Max Phase**)

  $v_i[p]$:= Compute $(v_i[p-1],\ Max)$

- Return $v_i[2f+2]$

Compute(t, *Function*) for node $i \in \mathcal{V}$

- $\tau_i := t$
- Perform $n-1$ rounds, each round consisting of the four steps below:

  Send τ_i to all the nodes in $N_i^+ \cup \{i\}$

  Receive values from $N_i^- \cup \{i\}$

  Denote the set of values received in the previous step as S_i

  $\tau_i := Function(S_i)$

- Return τ_i

Note that Algorithm Min-Max may execute for less number of rounds by exploiting the knowledge of topology. For simplicity, we require Algorithm Min-Max to execute for $n-1$ rounds. In [22], we show that under certain assumptions, usage of both Min and Max functions is necessary.

Correctness of Algorithm Min-Max with Binary Inputs.

We first prove a useful lemma, and introduce the notion of *source* of the graph.

LEMMA 2. *Suppose that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS. For any $F \subseteq \mathcal{V}$, such that $|F| \leq f$, let G_F denote the subgraph of G induced by the nodes in $\mathcal{V} - F$. There exists at least one node in G_F that has directed paths in G_F to all nodes in $\mathcal{V} - F$. Such a node is said to be a source for G_F.*

PROOF. The proof of the lemma is by contradiction. Suppose that Graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS, and for some $F \subseteq \mathcal{V}$, $|F| \leq f$, there exists a pair of nodes $i, j \notin F$ such that there is no node s that has directed paths to both i and j in subgraph G_F induced by nodes in $\mathcal{V} - F$. For the subgraph G_F and a node x in $\mathcal{V} - F$, define S_x as the set of all nodes that have directed paths in G_F to node x. Note that S_x contains x as well, because x trivially has a path to itself.

By assumption, S_i and S_j are disjoint. Moreover, there must be no path from any node in S_i to any node in S_j in G_F, and vice versa, since otherwise, there would exist some

node that can reach both nodes i and j, which contradicts our assumption above. Now, define L, C, R as follows: (i) $L := S_i$, (ii) $R := S_j$, and (iii) $C := \mathcal{V} - F - L - R$. We make the following observations:

- *F and C may be empty, but L and R are non-empty*: This is true because $i \in S_i = L$ and $j \in S_j = R$.
- *Nodes in C (if non-empty) have no link to nodes in $L \cup R$*: If some node $c \in C$ has a link to some node $x \in L = S_i$, then c will be able to reach node i on a directed path via node x (since $x \in S_i$ has a path to i, by definition of set S_i). This would then imply that c must be in S_i, however, that contradicts the definition of C as $\mathcal{V} - F - L - R$. By a similar argument, nodes in C cannot have links to nodes in R.
- *There is no link from any node in L to nodes in R, and vice versa*: Recall that $L = S_i$ and $R = S_j$. If some node $x \in L$ has a link to a node $y \in R$, then x will have a directed path to node j via node y. However, this contradicts our assumption above that no node has directed paths to both i and j.

These observations together imply that $L \cup C \overset{1}{\not\Rightarrow} R$ and $C \cup R \overset{1}{\not\Rightarrow} L$. That is, $G(\mathcal{V}, \mathcal{E})$ does not satisfy Condition CCS. This is a contradiction. Thus, Lemma 2 is proved. □

The lemma defines the notion of a *source* node. Essentially, the lemma shows the existence of a directed rooted spanning tree in the induced graph G_F, which has the source node as the root. Presence of such "source" nodes (or root) is crucial in achieving consensus. Similar observations were first made in the context of fault-free consensus [5, 13], and also in the context of other versions of fault-tolerant consensus problems [8, 24, 23, 6] (although the exact manner in which the source node is identified differs for the different problems). The key distinguishing feature of the results presented in this paper is in the algorithms developed to solve the consensus problems. For instance, the structure of the above Min-Max algorithm – making alternating use of Min and Max function – has not been applied for consensus previously, to the best of our knowledge.

LEMMA 3. *Algorithm Min-Max satisfies termination, agreement and validity properties.*

PROOF. The proof of correctness assumes that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS. Since Algorithm Min-Max executes a fixed number of phases, its termination occurs in finite time. Validity is satisfied trivially as well. Now we prove that the algorithm satisfies the agreement property when the inputs are binary (0 or 1). We start by observing that $Compute(t, Min)$ never returns a value larger than parameter t passed to *Compute*, and $Compute(t, Max)$ never returns a value smaller than parameter t passed to *Compute*.

Fix an execution of the algorithm. Since there are $2f + 2$ phases. There must exist a pair of consecutive phases p^*, p^*+1 such that *no node crashes* in phases p^* and p^*+1. Without loss of generality, let p^* be the Min Phase (i.e., p^* *mod* $2 = 0$) and $p^* + 1$ be the Max Phase.

Denote by F the set of nodes that crash before starting phase p^*. Lemma 2 shows the existence of a source node that has directed paths in G_F to all nodes in $\mathcal{V} - F$ (G_F is defined in the Lemma). In general, there may be multiple such source nodes in G_F. Consider the two cases below. In each case, we show that agreement is achieved.

- *Case I: There exists a source s in G_F for which $v_s[p^* - 1] = 0$*: Thus, during the Min Phase p^*, node s will call *Compute(0,Min)*. Then during the first round of *Compute* in phase p^*, those nodes in $\mathcal{V} - F$ with incoming links from node s will update their τ variable (within *Compute*) to be 0. Recall that we are presently assuming binary inputs. Since the source node has directed paths (of length at most $n - 1$) to all the nodes in G_F, it follows by induction that each node i in $\mathcal{V} - F$ will update its state τ_i to be 0 by the end of the $n - 1$ rounds performed within *Compute*. Thus, when *Compute* returns, $v_i[p^*]$ at each $i \in \mathcal{V} - F$ will be set to 0. It should be easy to see that the remaining phases will not change the value of v_i at the fault-free nodes, ensuring agreement when the algorithm terminates.
- *Case II: For each source s in G_F, $v_s[p^* - 1] = 1$*: In this case, we argue that, for each source s, $v_s[p^*] = 1$. Suppose, by way of contradiction, that each source s of G_F has $v_s[p^* - 1] = 1$, but there exists a source node s' for which $v_{s'}[p^*] = 0$. For this to happen, node s' must receive 0 on a path from some other *non-source* node z during phase p^*. This implies that $v_z[p^* - 1] = 0$; additionally, the fact that there exists a path in G_F from z to the source node s' implies that z is also a source in G_F. This contradicts the assumption that all source nodes in G_F have state equal to 1 at the start of phase p^*.

  This shows that, for each source node s, we have $v_s[p^*] = 1$. Now consider Max Phase $p^* + 1$. Recall that no node crashes in Phases p^* and $p^* + 1$. Thus, by an argument analogous to that used for Min Phase p^* in Case I above, it follows that, for all $i \in \mathcal{V} - F$, $v_i[p^* + 1] = 1$, achieving agreement. Any additional phases beyond phase $p^* + 1$ will not result in violation of the agreement, similar to Case I.

□

4.2.2 Multi-Valued Consensus

Algorithm Min-Max is proven correct for binary inputs. We will prove that the Algorithm MVC presented below performs exact consensus with inputs being some integer in the range $[0, K]$, where $K \geq 1$. Algorithm MVC uses Algorithm *Min-Max* as well as *Compute* defined above.

Algorithm MVC for node i

Initialization: $w_i[0] :=$ input of node i

Repeat for $l := 0$ to K

1. $w_i[l + 1] := Compute(w_i[l], Max)$
2. if $w_i[l] = l$, then $y_i[l] := 0$; otherwise, $y_i[l] := 1$
3. if Min-Max($y_i[l]$) returns 0, then
   terminate with output l

Step 3 of the algorithm performs Algorithm Min-Max, with node i passing $y_i[l]$ as the parameter to Min-Max. The

"if" statement in step 3 checks the value returned by Algorithm Min-Max, and terminates the algorithm if the returned value is 0. Recall that the system is synchronous. Thus, all fault-free nodes perform *Min-Max* in step 3 in iteration l of the algorithm synchronously. Similarly, all fault-free nodes perform *Compute* (in step 1) in iteration l of the algorithm synchronously.

Correctness of Algorithm MVC.

The proof assumes that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS. Observe that Algorithm MVC specifies up to $K+1$ iterations, each iteration consisting of 3 steps enumerated in the algorithm. However, the algorithm may potentially terminate before completing all $K+1$ iterations (due to the check in step 3).

Since Min-Max is a binary consensus algorithm, each node that completes execution of Algorithm Min-Max in iteration l of MVC (i.e., without crashing before this instance of Min-Max) will obtain the same return value from Min-Max in iteration l. This ensures that all fault-free nodes terminate after completing an identical number of iterations. Now we present a lemma that is useful to prove correctness of Algorithm MVC.

LEMMA 4. *If a node i initiates iteration l of Algorithm MVC ($0 \leq l \leq K$), then (i) $w_i[l] \geq l$, and (ii) $w_i[l]$ equals the input of some node in the system.*

PROOF. The proof is by induction. Consider $l = 0$. $w_i[0]$ is initialized to be the input of node i. All inputs are in the range $[0, K]$, which implies that $w_i[0] \geq 0$. Thus, both parts of the statement of the lemma are true for $l = 0$.

Now assume that the statement of the lemma is true for some l, where $0 \leq l < K$. Thus, for each node i that initiates iteration l, $w_i[l] \geq l$ and $w_i[l]$ is the input of some node in the system. Now, if Algorithm MVC terminates without performing iteration $l+1$, the proof of the lemma is complete. Therefore, let us now consider the case that Algorithm MVC is not terminated after completing l iterations. This implies that Min-Max($y_i[l]$) returns 1 in Step 3 of iteration l. Since Algorithm Min-Max is a binary consensus algorithm, the validity condition and the fact that it returns 1 implies that there exists some node j with $y_j[l] = 1$ such that (i) j did not crash before starting the execution of Min-Max($y_j[l]$) in step 3 of iteration l at node j, and (ii) node i has a path from node j consisting only of nodes that have *not* crashed before initiating step 3 of iteration l. Due to the rules for setting the value of $y_j[l]$ in step 2, and the fact that $y_j[l] = 1$, we can infer that that $w_j[l] > l$.

Finally, all nodes that are fault-free at least until initiating step 3 in iteration l must also be fault-free until they finish performing step 1 in iteration l. This implies that when *Compute* is performed in step 1 of iteration l, there exists a path from node j to node i. Existence of this path then ensures that, at node i, *Compute* returns a value $\geq w_j[l] > l$. Thus, $w_i[l+1] > l$. That is, $w_i[l+1] \geq l+1$.

Secondly, *Compute* at node i in iteration l will only return a value that is passed as the first parameter to *Compute* by at least one of the nodes initiating the execution of *Compute* in iteration l. Thus, $w_i[l+1]$ must equal $w_j[l]$ for some node j – by induction basis, $w_j[l]$ must be the input of some node in the syatem, and therefore, $w_i[l+1]$ also equals the input of that node. This concludes the proof. □

LEMMA 5. *Algorithm MVC satisfies termination, agreement and validity properties.*

PROOF. Termination in finite time occurs because Algorithm *Min-Max* and *Compute* return in finite time, and Algorithm MVC performs a finite number of iterations. As discussed earlier, all fault-free nodes terminate after completing an *identical* number of iterations. There are two cases to consider depending on the number of iterations completed before the algorithm is terminated.

Case 1: Algorithm MVC terminates only after completing the last iteration with $l = K$. Due to Lemma 4, for each node j that initiates iteration with $l = K$, we have $w_j[K] \geq K$. However, since all inputs are in the range $[0, K]$, and due to the use of *Compute* to update w_j, we have that $w_j[K] \leq K$ as well. This means that $w_j[K] = K$. But by Lemma 4, $w_j[K]$ is also the input of some node in the system; it then follows that some node in the system had value K as the input.

Since $w_j[K] = K$, in iteration $l = K$, $y_j[K] = 0$ for each node j that completes step 2, and thus Min-Max can only return 0 in step 3, resulting in the output value of K at all fault-free nodes. Thus agreement is achieved. As argued above, K is the input of some node, the validity property is also satisfied.

Case 2: Algorithm MVC terminates after completing iteration $l < K$. This implies that all nodes that are fault-free until the end of iteration l must have necessarily obtained the return value of 0 from Min-Max in iteration l, and therefore, they will all terminate with output value l. Thus agreement is achieved. The fact that the return value from Min-Max is 0 implies that at least one node, say node j, must have initiated Min-Max in iteration l with $y_j[l] = 0$, implying (due to step 2) that $w_j[l] = l$. By Lemma 4, $w_j[l] = l$ must be the input of some node. Since l is also the output of the algorithm in this case, validity condition is also satisfied. □

5. APPROXIMATE CRASH-TOLERANT ASYNCHRONOUS CONSENSUS

Approximate crash-tolerant consensus must satisfy the following three properties, where $\epsilon > 0$: (i) *ϵ-agreement*: the difference between outputs at any two fault-free nodes is $< \epsilon$. (ii) *Validity*: the output at any fault-free node is within the range of the inputs at all the nodes. (iii) *Termination:* every fault-free node decides on an output in finite amount of time. For simplicity, we assume that the input at each node is some *real number* in the range $[0, K]$. Note that if $K < \epsilon$, then the problem is trivial, so K is assumed to be $\geq \epsilon$.

Theorem 2 in Section 3 presents the necessary and sufficient condition (named Condition CCA) for solving the above problem in directed graphs.

5.1 Necessity of Condition CCA

LEMMA 6. *Condition CCA is necessary for approximate crash-tolerant consensus in an asynchronous system.*

PROOF. The proof is by contradiction. Suppose that there exists a correct approximate consensus algorithm in $G(\mathcal{V}, \mathcal{E})$, but $G(\mathcal{V}, \mathcal{E})$ does not satisfy the condition in the lemma. That is, there exists a node partition L, C, R such that L and R are non-empty, and $L \cup C \overset{f+1}{\not\Rightarrow} R$ and $C \cup R \overset{f+1}{\not\Rightarrow} L$.

Let $O(L)$ denote the set of nodes in $C \cup R$ that have outgoing links to nodes in L, i.e., $O(L) = \{i \mid i \in C \cup R, \ \ N_i^+ \cap L \neq \emptyset\}$. Similarly, define $O(R) = \{j \mid j \in L \cup C, \ \ N_j^+ \cap R \neq \emptyset\}$. Since $L \cup C \overset{f+1}{\not\Rightarrow} R$ and $C \cup R \overset{f+1}{\not\Rightarrow} L$, we have $|O(L)| \leq f$ and $|O(R)| \leq f$.

Consider the scenario where (i) each node in L has input 0; (ii) each node in R has input ϵ; (iii) nodes in C (if non-empty) have arbitrary inputs in $[0, \epsilon]$; (iv) no node crashes; and (v) the message delay for communications channels from $O(L)$ to L and from $O(R)$ to R is arbitrarily large compared to all the other channels. Recall that such a scenario is possible, since we have assumed that the input range is $[0, K]$, where $K \geq \epsilon$. Consider nodes in L.

Since messages from the set $O(L)$ take arbitrarily long to arrive at the nodes in L, and $|O(L)| \leq f$, from the perspective of the nodes in L, the nodes in $O(L)$ appear to have crashed. Thus, nodes in L must decide on their output without waiting to hear from the nodes in $O(L)$. Consequently, to satisfy the validity property, the output at each node in L has to be 0, since 0 is the input of all the nodes in L. Similarly, nodes in R must decide their output without hearing from the nodes in $O(R)$; they must choose output as ϵ, because the input at all the nodes in R is ϵ. Thus, the ϵ-agreement property is violated, since the difference between outputs at fault-free nodes is not $< \epsilon$. This is a contradiction. □

5.2 Sufficiency of Condition CCA

We prove the sufficiency of Condition CCA constructively by presenting an algorithm, called Algorithm WA (Wait-and-Average), and proving its correctness. The algorithm, presented below, assumes that each node has the knowledge of the network topology, and the algorithm proceeds in *asynchronous* phases. In each phase, nodes flood messages containing the current value of their state variable v, their identifier, and a phase index. Each node i waits until it has received an "adequate" set of values from other nodes, where "adequate" is made precise by Condition WAIT defined below. Then, node i updates its state variable v to be the *average* of set of values received in the current phase, and then proceeds to the next phase. When node i has finished p_{end} phases, it produces an output that equals the current value of state variable v; p_{end} is an integer $> \log_{n/(n-1)} \frac{K}{\epsilon}$.

In Algorithm WA, observe that $heard_i[p]$ is the set of nodes from which node i has received values during phase p. As seen in the algorithm pseudo-code, node i performs the averaging operation to update its state variable v_i when Condition WAIT below holds for the first time. Algorithm WA is an extension of an approximate consensus algorithm for complete graphs [10]. The key contribution of our work is to identify *Condition WAIT*.

Condition WAIT: For $F_i \subseteq \mathcal{V}$, where $|F_i| \leq f$, denote by $reach_i(F_i)$ the set of nodes that have paths to node i in the subgraph induced by the nodes in $\mathcal{V} - F_i$ (i.e., the paths do not contain nodes in F_i). Condition *WAIT* is satisfied at node i if *there exists* a set $F_i \subseteq \mathcal{V}$, where $|F_i| \leq f$, such that $reach_i(F_i) \subseteq heard_i[p]$. ($reach_i(F_i)$ may be different in each phase, since it depends on the message delays. For simplicity, we ignore the phase index p in the notation.)

Algorithm WA for node $i \in \mathcal{V}$

---

p_{end} is an integer $> \log_{n/(n-1)} \frac{K}{\epsilon}$.

- $v_i[0] :=$ input at node i
- For Phase $p := 1$ to p_{end}:
  - On entering phase p:
    $R_i[p] := \{v_i[p-1]\}$
    $heard_i[p] := \{i\}$
    Send message $(v_i[p-1], i, p)$ to
    all the outgoing neighbors
  - When message (h, j, p) is received for the *first time*:
    $R_i[p] := R_i[p] \cup \{h\}$ // $R_i[p]$ is a multiset
    $heard_i[p] := heard_i[p] \cup \{j\}$
    Send message (h, j, p) to
    all the outgoing neighbors
  - When Condition *WAIT* holds for the first time in phase p:
    $v_i[p] := \frac{\sum_{v \in R_i[p]} v}{|R_i[p]|}$
    If $p < p_{end}$, begin the next phase
    Else, output v_i

Note that $R_i[p]$ is a multiset, and thus may contain multiple instances of the same value.

Correctness of Algorithm WA.

We first prove a useful lemma. Let $heard_i^*[p]$ denote the set $heard_i[p]$ when Condition WAIT holds for the first time at node i in phase p. The correctness of Algorithm WA relies on the following lemma, which assumes that graph $G(\mathcal{V}, \mathcal{E})$ satisfies condition CCA. In a given execution, define $F[p]$ as the nodes that have *not* computed value $v[p]$ in phase p, i.e., nodes in $F[p]$ have crashed before computing $v[p]$.

LEMMA 7. *For phase $p \geq 1$, consider two nodes $i, j \in \mathcal{V} - F[p]$. Then, $heard_i^*[p] \cap heard_j^*[p] \neq \emptyset$.*

PROOF. By construction, there exist two sets F_i and F_j such that Condition WAIT holds for sets $heard_i^*[p]$ and F_i at node i, and for sets $heard_j^*[p]$ and F_j at node j. In other words, (i) $F_i \subseteq \mathcal{V}$ and $|F_i| \leq f$, (ii) $F_j \subseteq \mathcal{V}$ and $|F_j| \leq f$, (iii) $reach_i(F_i) \subseteq heard_i^*[p]$, and (iv) $reach_j(F_j) \subseteq heard_j^*[p]$. If $reach_i(F_i) \cap reach_j(F_j) \neq \emptyset$, then the proof is complete, since $reach_i(F_i) \subseteq heard_i^*[p]$ and $reach_j(F_j) \subseteq heard_j^*[p]$. Thus, $heard_i^*[p] \cap heard_j^*[p] \neq \emptyset$.

Now, consider the case when $reach_i(F_i) \cap reach_j(F_j) = \emptyset$. We will derive a contradiction in this case. Recall that $reach_i(F_i)$ is defined as the set of nodes that have paths to node i in the subgraph induced by the nodes in $\mathcal{V} - F_i$. This implies that in graph G, the incoming neighbors of set $reach_i(F_i)$ are contained in set F_i. Similarly, in graph G, the incoming neighbors of set $reach_j(F_j)$ are contained in set F_j.

In graph G, we will find subsets of nodes L, C, R that violate Condition CCA. Let $L = reach_i(F_i)$, $R = reach_j(F_j)$ and $C = \mathcal{V} - L - R$. Observe that since $reach_i(F_i) \cap reach_j(F_j) = \emptyset$, L, C, R form a partition of $\mathcal{V}$. Moreover, $i \in reach_i(F_i)$ and $j \in reach_j(F_j)$; hence, $L = reach_i(F_i)$

and $R = reach_j(F_j)$ are both non-empty. Let $N(L)$ be the set of incoming neighbors of set L. By definition, $N(L)$ is contained in $R \cup C$. Since $L = reach_i(F_i)$, the only nodes that may be in $N(L)$ are also in F_i as argued above, i.e., $N(L) \subseteq F_i$. By assumption, $|F_i| \leq f$. Therefore, $|N(L)| \leq f$, which implies that $R \cup C \overset{f+1}{\not\Rightarrow} L$. Similarly, we can argue that $L \cup C \overset{f+1}{\not\Rightarrow} R$. These two conditions together show that G violates Condition CCA, a contradiction. Thus, $reach_i(F_i) \cap reach_j(F_j) \neq \emptyset$, which implies that $heard_i^*[p] \cap heard_j^*[p] \neq \emptyset$. This completes the proof. □

LEMMA 8. *Algorithm MVC satisfies termination, ϵ-agreement and validity properties.*

PROOF. Validity is trivially true. Denote by F the set of faulty nodes in an execution. Then, observe that Condition WAIT eventually holds at each fault-free node i in each phase $p \leq p_{end}$, because by choosing $F_i = F$, $reach_i(F) \subseteq heard_i[p]$ eventually. This together with the fact that p_{end} is bounded imply termination. Now, we prove that Algorithm WA achieves ϵ-agreement. The methodology in this proof is borrowed from other related work (e.g., [10, 3, 17, 5]).

Recall that $F[p]$ is defined as the nodes that have *not* computed value $v[p]$ in phase p, i.e., nodes in $F[p]$ have crashed before computing $v[p]$. Thus, $\mathcal{V} - F[p]$ is the set of nodes that complete the computation of $v[p]$. In the discussion below, let us denote by $\|x\|$ the absolute value of a real number x. For $p \geq 1$, define

$$M[p] = \max_{i \in \mathcal{V} - F[p]} v_i[p] \quad \text{and} \quad m[p] = \min_{i \in \mathcal{V} - F[p]} v_i[p] \quad (1)$$

With a slight abuse of terminology, let $M[0]$ and $m[0]$ denote the upper bound and the lower bound on the inputs, respectively. Define $\psi[p]$ as the maximum difference between states at nodes in $\mathcal{V} - F[p]$ in the end of phase p. Thus,

$$\psi[p] = \max_{i,j \in \mathcal{V} - F[p]} \|v_i[p] - v_j[p]\| = M[p] - m[p] \quad (2)$$

By definition of $m[p-1]$ and $M[p-1]$, for each node $k \in \mathcal{V} - F[p-1]$, we have

$$m[p-1] \leq v_k[p-1] \leq M[p-1] \quad (3)$$

$$m[p-1] \leq v_k[p-1] \leq M[p-1] \quad (4)$$

Now consider nodes $i, j \in \mathcal{V} - F[p]$. The multiset $R_i[p]$ at node i changes when node i receives new messages. Let $R_i^*[p]$ denote the multiset $R_i[p]$ used by node i to compute $v_i[p]$. Similarly, let $R_j^*[p]$ denote the multiset $R_j[p]$ used by node j to compute $v_j[p]$.

Let $r_i = |R_i^*[p]|$, the size of the multiset $R_i^*[p]$. Similarly, let $r_j = |R_i^*[p]|$. For brevity, the notation r_i and r_j does not include the phase index $[p]$.

By Lemma 7, there exists a common value in $R_i^*[p]$ and $R_j^*[p]$. Denote by c the common value. Note that by construction c is a state of some node in $\mathcal{V} - F[p-1]$. Thus, $m[p-1] \leq c \leq M[p-1]$. Define

$$\gamma = \frac{1}{n}.$$

By Algorithm WA, we have

$$\begin{aligned} v_i[p] &= \sum_{k \in R_i^*[p]} \frac{1}{r_i} v_k[p-1] \\ &\leq \frac{c}{r_i} + (1 - \frac{1}{r_i}) M[p-1] \quad \text{due to (1)} \\ &\leq \gamma c + (\frac{1}{r_i} - \gamma) c + (1 - \frac{1}{r_i}) M[p-1] \\ &\leq \gamma c + (1-\gamma) M[p-1] \end{aligned} \quad (5)$$

The last inequality is because $\gamma \leq \frac{1}{r_i}$ and $c \leq M[p-1]$.

$$\begin{aligned} v_j[p] &= \sum_{k \in R_j^*[p]} \frac{1}{r_j} v_k[p-1] \\ &\geq \frac{c}{r_j} + (1 - \frac{1}{r_j}) m[p-1] \quad \text{due to (1)} \\ &\geq \gamma c + (\frac{1}{r_j} - \gamma) c + (1 - \frac{1}{r_j}) m[p-1] \\ &\geq \gamma c + (1-\gamma) m[p-1] \end{aligned} \quad (6)$$

The last inequality is because $\gamma \leq \frac{1}{r_j}$ and $c \geq m[p-1]$. Now, subtracting (6) from (5), we get

$$v_i[p] - v_j[p] \leq (1-\gamma)(M[p-1] - m[p-1]) \quad (7)$$

By swapping the role of i and j above, we can show that

$$v_j[p] - v_i[p] \leq (1-\gamma)(M[p-1] - m[p-1]) \quad (8)$$

(7) and (8) together imply that

$$\begin{aligned} \|v_i[p] - v_j[p]\| &\leq (1-\gamma)(M[p-1] - m[p-1]) \\ &\leq (1-\gamma)\psi[p-1] \qquad \text{due to} \quad (2) \end{aligned}$$

Note that the first inequality is because $M[p-1] \geq m[p-1]$. The above inequality holds for each pair of nodes i, j that have computed $v[p]$ in phase p, so we have

$$\max_{i,j \in \mathcal{V} - F[p]} \|v_i[p] - v_j[p]\| \leq (1-\gamma)\psi[p-1]$$

This together with (2) implies that

$$M[p] - m[p] \leq (1-\gamma)(M[p-1] - m[p-1]) \quad (9)$$

By repeated application of (9), we get

$$M[p] - m[p] \leq (1-\gamma)^p (M[0] - m[0]) \quad (10)$$

Therefore, for a given $\epsilon > 0$, if

$$p > \log_{1/(1-\gamma)} \frac{M[0] - m[0]}{\epsilon}, \quad (11)$$

then

$$M[p] - m[p] < \epsilon \quad (12)$$

Recall that we have assumed that the input range is $[0, K]$, where $K \geq \epsilon$. Thus, $M[0] \leq K$ and $m[0] \geq 0$. Also, $\gamma = \frac{1}{n}$ by definition. Then, if we choose

$$p_{end} > \log_{n/(n-1)} \frac{K}{\epsilon},$$

then Algorithm WA satisfies ϵ-agreement property due to (11) and (12). □

6. EXACT BYZANTINE CONSENSUS IN SYNCHRONOUS SYSTEMS

An exact Byzantine consensus algorithm with *binary* inputs must satisfy the following properties: (i) *Agreement*: the output (i.e., decision) at all the fault-free nodes is identical. (ii) *Validity*: the output of every fault-free node equals the input of a fault-free node. (iii) *Termination:* every fault-free node decides on an output in finite amount of time. For multi-valued Byzantine consensus, we consider the weak version of validity [14]: If all fault-free nodes have the same input, then the output of every fault-free node equals its input. Theorem 3 in Section 3 presents the necessary and sufficient condition (named Condition BCS) for solving the above problem in directed graphs.

Due to lack of space, the proofs and the algorithms are presented in [21]. We show [21] that BCS is necessary using the state-machine approach [11, 9]. To prove sufficiency, we first develop an algorithm to achieve Byzantine consensus with *binary* inputs, and then use it to achieve multi-valued Byzantine consensus. We now introduce the notion of *reduced graph* and then present an observation that is useful to construct a Byzantine consensus algorithm in directed graphs that satisfy Condition BCS.

DEFINITION 2. **(Reduced Graph)** *For a given graph $G(\mathcal{V},\mathcal{E})$, and sets $F \subset \mathcal{V}$, $F_1 \subset \mathcal{V}-F$, such that $|F| \leq f$ and $|F_1| \leq f$, reduced graph $G_{F,F_1}(\mathcal{V}_{F,F_1},\mathcal{E}_{F,F_1})$ is defined as follows: (i) $\mathcal{V}_{F,F_1} = \mathcal{V}-F$, and (ii) $\mathcal{E}_{F,F_1}$ is obtained by removing from $\mathcal{E}$ all the links incident on the nodes in F, and all the outgoing links from nodes in F_1. That is, $\mathcal{E}_{F,F_1} = \mathcal{E} - \{(i,j) \mid i \in F \text{ or } j \in F\} - \{(i,j) \mid i \in F_1\}$.*

Observation: Suppose $G(\mathcal{V},\mathcal{E})$ satisfies Condition BCS. Then, for any $F \subset \mathcal{V}$ and $F_1 \subset \mathcal{V}-F$, where $|F| \leq f$ and $|F_1| \leq f$, there exists a set of at least $f+1$ nodes $S \subseteq \mathcal{V}-F$ such that (i) nodes in S are strongly connected in reduced graph G_{F,F_1}, and (ii) for each $j \in \mathcal{V}-F-S$, there exist at least $f+1$ pairwise node-disjoint paths from S to j in G that do not contain any nodes in F.

The proposed algorithm for binary consensus maintains a state variable at each node, with the invariant that this state variables at each fault-free node aways has a "valid" value, where a value is "valid" if it is an input at some fault-free node. Intuitively, the above $f+1$ disjoint paths from the nodes in S to nodes in $\mathcal{V}-F-S$ provide adequate redundancy to allow propagation of values from the nodes in S to the nodes in $\mathcal{V}-F-S$, with the guarantee that any potential message tampering by faulty nodes would not cause the recipients to accept an "invalid" value. In the fortuitous event that the nodes in F are faulty and the nodes in S (which are fault-free) have the same valid state variable, this value is then propagated to all the fault-free nodes, achieving consensus. The algorithm ensures that this fortuitous event occurs at least once during the execution. The rest of the details are presented in [21] for lack of space.

7. DISCUSSION

As noted in Section 3, Condition CCS, CCA, and BCS capture how information can "flow between" different subsets of fault-free nodes despite the presence of faulty nodes under different synchrony assumptions. This section compares the three conditions and the condition for undirected graphs identified in [9, 11].

Comparison of Condition CCS, CCA, and BCS.

LEMMA 9. *Conditions CCS, CCA and BCS are progressively stronger. In particular, (i) Condition BCS implies Condition CCA, but not vice-versa, and (ii) Condition CCA implies Condition CCS, but not vice-versa.*

PROOF.

- Proof of claim (i):

  It should be easy to see that Condition CCA can be viewed as a special case of Condition BCS, if we force set F in Condition BCS to be an empty set. Therefore, Condition BCS implies Condition CCA.

  A clique consisting of $2f+1$ nodes satisfies Condition CCA but not Condition BCS; thus, (not surprisingly) Condition CCA does not imply BCS.

- Proof of claim (ii):

  We will prove that CCA implies CCS by contradiction. Suppose that graph G does not satisfy Condition CCS, i.e., there exists a node partition F, L, C, R of $\mathcal{V}$ such that $L \cup C \overset{1}{\not\Rightarrow} R$ and $R \cup C \overset{1}{\not\Rightarrow} L$. Then, define $C' = C \cup F$. Due to the fact that $|F| \leq f$, $L \cup C' \overset{f+1}{\not\Rightarrow} R$ and $R \cup C' \overset{f+1}{\not\Rightarrow} L$, violating Condition CCA. This proves that CCA implies CCS.

  Consider the example network in Figure 2 in Section 3. This network tolerates 1 crash fault in synchronous systems, since it satisfies Condition CCS; however, it does not satisfy Condition CCA when $L = \{v_1\}, C = \emptyset$ and $R = \{v_2, v_3\}$. Thus, CCS does not imply CCA.

□

Comparison of Conditions in Undirected and Directed Graphs.

Previously necessary conditions for undirected graphs [11, 9, 3, 17] all imply that each pair of fault-free nodes can communicate *reliably* despite the presence of f faulty nodes. In particular, this is true due to $f+1$ connectivity in case of crash faults, and $2f+1$ connectivity for Byzantine faults. Specifically, $2f+1$-connectivity implies the presence of $2f+1$ node-disjoint paths between nodes that are not neighbors, allowing each pair of nodes to communicate reliably despite f Byzantine faulty nodes. In contrast, in *directed graphs*, to be able to achieve consensus, it is *not necessary* for **all** node pairs to be able to communicate with each other reliably (even in just one direction). This is a manifestation of the "asymmetry" noted in our discussion above. For instance, the network in Figure 2 can tolerate 1 crash fault in a synchronous setting; however, v_3 does not have paths to the other nodes. Now, consider a network that consists of 6 nodes, 4 of which (say, w_1, w_2, w_3, w_4) form a clique, and also have directed links to nodes w_5 and w_6. Node w_5 has no path to w_6 and vice versa. Yet, Byzantine consensus can be achieved easily by first reaching consensus within the 4-node clique, and then propagating the consensus value (for the 4-node consensus) to nodes w_5 and w_6. Nodes w_5 and w_6 can choose majority of the values received from the nodes in the 4-node clique as its own output. It should be easy to see that this algorithm works correctly for inputs

in $\{0, 1\}$ as required in the Byzantine consensus formulation considered in this work. However, nodes w_5 and w_6 cannot communicate reliably with each other (in either direction).

There exist graphs that satisfy Condition BCS wherein reliable communication may not be feasible in either direction across a given cut. In particular, consider the network in Figure 1 in Section 3 again. Observe that there are only 4 directed links from K_1 to K_2, and 4 directed links from K_2 to K_1. Thus, reliable communication is *not guaranteed* across the cut (K_1, K_2) in either direction when $f = 2$ (Byzantine faults). Yet, Byzantine consensus is achievable in synchronous systems since this graph satisfies Condition BCS for $f = 2$. We prove this claim in [21].

8. SUMMARY

Necessary and sufficient conditions for solving the following problems in *directed* graphs are presented: (i) exact crash-tolerant consensus in *synchronous* systems, (ii) approximate crash-tolerant consensus in *asynchronous* systems, and (iii) exact Byzantine consensus in *synchronous* systems. Development of efficient algorithms for solving these problems is a topic for future work. Also, tight condition for approximate Byzantine asynchronous consensus in directed graphs remains unknown.

9. REFERENCES

[1] I. Abraham, Y. Amit, and D. Dolev. Optimal resilience asynchronous approximate agreement. In *OPODIS*, pages 229–239, 2004.

[2] E. Alchieri, A. Bessani, J. Silva Fraga, and F. Greve. Byzantine consensus with unknown participants. In T. Baker, A. Bui, and S. Tixeuil, editors, *Principles of Distributed Systems*, volume 5401 of *Lecture Notes in Computer Science*, pages 22–40. Springer Berlin Heidelberg, 2008.

[3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley Series on Parallel and Distributed Computing, 2004.

[4] P. Bansal, P. Gopal, A. Gupta, K. Srinathan, and P. K. Vasishta. Byzantine agreement using partial authentication. In *Proceedings of the 25th international conference on Distributed computing*, DISC'11, pages 389–403, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Optimization and Neural Computation Series. Athena Scientific, 1997.

[6] M. Biely, P. Robinson, and U. Schmid. Agreement in directed dynamic networks. In *Structural Information and Communication Complexity*, volume 7355 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2012.

[7] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *CoRR*, abs/1408.0620, 2014.

[8] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks. *CoRR*, abs/1408.0620, 2014.

[9] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, March 1982.

[10] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33:499–516, May 1986.

[11] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, PODC '85, pages 59–70, NY, USA, 1985. ACM.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, April 1985.

[13] A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *Automatic Control, IEEE Transactions on*, 48(6):988–1001, June 2003.

[14] L. Lamport. The weak Byzantine generals problem. *J. ACM*, 30(3):668–676, July 1983.

[15] H. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram. Resilient asymptotic consensus in robust networks. *IEEE Journal on Selected Areas in Communications: Special Issue on In-Network Computation*, 31:766–781, April 2013.

[16] H. LeBlanc, H. Zhang, S. Sundaram, and X. Koutsoukos. Consensus of multi-agent networks in the presence of adversaries using only local information. *HiCoNs*, 2012.

[17] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[18] A. Maurer, S. Tixeuil, and X. Défago. Reliable communication in a dynamic network in the presence of Byzantine faults. *CoRR*, abs/1402.0121, 2014.

[19] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, Apr. 1980.

[20] L. Su and N. Vaidya. Reaching approximate Byzantine consensus with multi-hop communication. *CoRR*, abs/1411.5282, 2014.

[21] L. Tseng and N. H. Vaidya. Exact Byzantine consensus in directed graphs. *CoRR*, abs/1208.5075, 2012.

[22] L. Tseng and N. H. Vaidya. Crash-Tolerant Consensus in Directed Graphs. *CoRR*, abs/1412.8532, 2014.

[23] L. Tseng and N. H. Vaidya. Iterative approximate Byzantine consensus under a generalized fault model. In *International Conference on Distributed Computing and Networking (ICDCN)*, January 2013.

[24] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate Byzantine consensus in arbitrary directed graphs. In *Proceedings of the thirty-first annual ACM symposium on Principles of distributed computing*, PODC '12. ACM, 2012.

[25] H. Zhang and S. Sundaram. Robustness of complex networks with implications for consensus and contagion. In *Proceedings of CDC 2012, the 51st IEEE Conference on Decision and Control*, 2012.

[26] H. Zhang and S. Sundaram. Robustness of distributed algorithms to locally bounded adversaries. In *Proceedings of ACC 2012, the 31st American Control Conference*, 2012.