Capacity of Byzantine Agreement with Finite Link Capacity

Guanfeng Liang and Nitin Vaidya
Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign, USA
Email: {gliang2,nhv}@illinois.edu

***Abstract*—We consider the problem of maximizing the throughput of Byzantine agreement, when communication links have finite capacity. Byzantine agreement is a classical problem in distributed computing. In existing literature, the communication links are implicitly assumed to have infinite capacity. The problem changes significantly when the capacity of links is finite. We define the throughput and capacity of agreement, and identify necessary conditions of achievable agreement throughputs. We propose an algorithm structure for achieving agreement capacity in general networks. We also introduce capacity achieving algorithms for two classes of networks: (i) arbitrary four-node networks with at most 1 failure; and (ii) symmetric networks of arbitrary size.**

I. Introduction

Byzantine agreement, with initial solutions presented in the seminal work of Pease, Shostak and Lamport [1], [2], is a classical problem in distributed computing and has been implemented [3], [4] and widely used in fault-tolerant distributed systems design. Despite having been well-studied in the literature for a wide variety of settings, little or no work has been done that takes into account the capacity of the communication network when implementing a Byzantine agreement algorithm. As a result, the communication network is typically underutilized by existing Byzantine agreement algorithms. We consider the problem of optimizing the **throughput** of Byzantine agreement. As we will see, in some networks, we can improve the throughput of agreement over existing algorithms by a factor linear in the size of the network (symmetric networks in Section V).

A. Byzantine Agreement

There are two flavors of the BA problem: consensus and broadcast. In this paper, we only discuss the broadcast version of the BA problem. The broadcast version of the BA problem considers a network with one node designated as the *sender* or *source* (S), and the other nodes designated as the *peers*. The goal of BA is for all the fault-free nodes to "agree on" the value being sent by the sender, despite the possibility that some of the nodes may be faulty. In particular, the following conditions must be satisfied:

- **Agreement:** All fault-free peers must agree on an identical value.
- **Validity:** If the sender is fault-free, then the agreed value must be identical to the sender's value.
- **Termination:** Agreement between fault-free peers is eventually achieved.

B. Models

1) Network Model: We assume a synchronous network modeled as a fully-connected directed graph. Each directed link is associated with a *fixed* capacity, which specifies the maximum amount of information that can be transmitted on that link per unit time. That is, in time t on a link with capacity c, ct bits can be sent. The capacity of some links may be 0, which implies that these links do not exist. This model is a special case of the *channel capacity* from the information theory literature [5], and has been widely used in the network coding literature [6], [7], [8], [9].

In the literature on Byzantine agreement, the capacity of communication links are implicitly assumed to be infinite. To our knowledge, we are the **first** to study the problem of Byzantine agreement when the links in the network have finite capacity.

2) Adversary Model: We assume that the adversary has complete knowledge of the network topology, the algorithm, the information the source is trying to send, and no secret is hidden from the adversary. The adversary can take over up to t processors at any point during the algorithm, where $t < n/3$. These nodes are said to be *faulty*. The faulty nodes can engage in any kind of deviations from the algorithm, including sending false messages, collusion, and crash failures.

C. Capacity of Agreement

Our goal in this work is to design algorithms that can achieve optimal *throughput* of agreement.

When defining throughput, the "value" referred in the above definition of agreement is viewed as an infinite sequence of *information* bits. We assume that the information bits have already been compressed, such that for any subsequence of length $l > 0$, the 2^l possible sequences are sent by the sender with equal probability. Thus, no set of information bits sent by the sender contains useful information about other bits. This assumption comes from the observation about "typical sequences" in Shannon's work [5].

At each peer, we view the agreed information as being represented in an array of infinite length. Initially, none of the bits in this array at a peer have been agreed upon. As time progresses, the array is filled in with agreed bits. In principle, the array may not necessarily be filled sequentially. For instance, a peer may agree on bit number 3 before it is able to agree on bit number 2. Once a peer agrees on any bit, that agreed bit cannot be changed.

We assume that an agreement algorithm begins execution at time 0. The system is assumed to be synchronous. In a given execution of an agreement algorithm, suppose that by time t all the fault-free peers have agreed upon bits 0 through $b(t)-1$, and at least one fault-free peer has not yet agreed on bit number $b(t)$. Then, the agreement *throughput* is defined as[1] $\lim_{t\to\infty} \frac{b(t)}{t}$.

Capacity of agreement in a given network, for a given sender and a given set of peers, is defined as the supremum of all achievable agreement throughputs.

Prior work [1], [2] shows that, to achieve agreement despite t failures in such networks, the number of nodes n must exceed $3t$. For such networks, our contributions are two-fold:

- We identify necessary conditions that can be used to obtain an upper bound on the agreement capacity of arbitrary networks. This is the first work that investigates the performance of Byzantine agreement algorithms under the realistic assumption that point-to-point links in the network are capacity-constrained.

- Although determining the capacity of general networks is very difficult, we are able to solve the problem for two classes of networks:
  - Due to the requirement that $n > 3t$, the smallest network wherein the agreement problem is meaningful consists of $n = 4$ nodes, with at most 1 failure. For such networks, we fully characterize the agreement capacity. The different four-node networks may differ in the capacity of different links in the network. We present a capacity achieving algorithm for four-node networks with *arbitrary* link capacity distribution.

  - For fully connected symmetric networks, in which all directed links have the same capacity C, with $n \geq 4$ nodes and up to $t < n/3$ faulty nodes, we show that the agreement capacity is $(n-t-1)C$. This is at least $n-t-1$ times higher than the throughput achievable with traditional agreement algorithms.

II. Necessary Conditions for General Networks

In this section, we introduce three necessary conditions for agreement throughput at rate R bits/unit time to be achievable in general networks. These necessary conditions together serve as an upper bound on the agreement capacity.

It is known that a network must contain at least $n = 3t+1$ nodes for agreement to be achievable with t Byzantine failures. Consider a synchronous network of $n \geq 4$ nodes, and at most $t < n/3$ of these nodes may be faulty. For such networks, we identify the following *necessary* conditions for achieving agreement throughput of R bits/unit time in general networks.

- **Necessary condition NC1:** If any t peers are removed from the network, the min-cut from the source to each remaining peer must be $\geq R$.
- **Necessary condition NC2:** The max-flow to each of the peers from the other peers, with the source and any $t-1$ peers removed from the network, must be $\geq R$.
- **Necessary condition NC3:** If any t nodes are removed from the network, the sum capacity of all links in both directions on any cut must be $\geq R$.

The proofs for these conditions can be found in our technical report [10]. The necessary conditions above together serve as an upper bound on the agreement capacity of general networks. However, whether this bound is tight or not remains an open problem in general. In fact, even characterizing the capacity of multicast with node failures, which is a degraded version of the Byzantine agreement problem with the source node being always fault-free, is an open problem in general, and has been solved only for a few small networks [11], [12].

Despite the difficulty in solving the agreement capacity problem in general, we are able to characterize the agreement capacity for: (i) four-node networks with $n = 4, t = 1$ and arbitrary link capacity (Section IV); and (ii) symmetric networks with arbitrary size n (Section V).

III. The Algorithm Structure in a Nutshell

In this section we present the main idea and the structure of our algorithms. Any Byzantine agreement algorithm can be viewed as an error correction network code (possibly with loops and feedback), which encodes the data with sufficient redundancy such that all peers will be able to correct any inconsistency of the data, introduced by up to t faulty nodes, including the data source (in multicast, only the intermediate nodes can introduce inconsistency). It is to be noted that any error correction code can be used for error detection at the same rate. As a result, if agreement throughput of R bits/unit time is achievable in a network, there must exist an error detection network code at rate R for the same network. Based on this observation, we propose the following capacity achieving algorithm structure for Byzantine agreement. The sequence of information bits are divided into *generations* of same size, and agreement are achieved one generation after another by performing the following stages.

A. Error Detecting Stage

In this stage, the data of the current generation is encoded and transmitted using an error detection network code that can detect any misbehavior by up to t nodes at rate R. If no failure is detected, every fault-free peer agrees on the data of

[1] As a technicality, we assume that the limit exists. The definitions here can be modified suitably to take into account the possibility that the limit may not exist.

this generation. Then a new generation is carried out using the same error detection network code. If any peer detects a failure, the full broadcast stage is performed.

B. Full Broadcast Stage

In the full broadcast stage, every node (including the source) broadcasts to all other nodes everything it has sent and received in the error detecting stage. This broadcast is made reliable using a traditional Byzantine agreement algorithm (for example, the one proposed by Pease, Shostak and Lamport [2]). From the broadcast content, the fault-free nodes learn some information about the location of the faulty nodes. In particular, some links adjacent to the faulty nodes will be identified. Then a new error detection network code at rate R, according to which no information is transmitted on the identified links. This network code can always be found because one of the two nodes attached to a identified link must be faulty, and a faulty node can always misbehave by sending nothing on its out-going links and pretend not receiving anything on its incoming links. Thus, agreement at rate R is achievable even if no data is transmitted on these links. Hence the desired network code must exist. Then a new generation of data is transmitted using this code.

As we will see later, in a network with at most t failures, a faulty node is identified once more than t links adjacent to it is identified. Once a faulty node is located, it is isolated by removing all links attached to it. As a result, after at most $t(t+1)$ failures have been detected, all faulty nodes will be removed from the network. Then we only need to perform a traditional multicast at rate R, or terminate the algorithm if the source is identified as faulty.

IV. Byzantine Agreement in Four-Node Networks

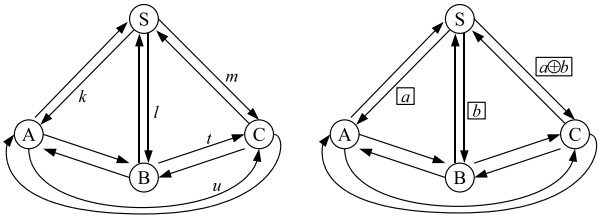
Consider the network of 4 nodes, named S, A, B, C, with node S acting as the source, and nodes A, B, C being the peers. For four-node networks, the following two conditions are also necessary.

- **Necessary condition NC4:** All incoming links to the peers must exist (capacity > 0).
- **Necessary condition NC5:** The capacity of every out-going link from S must be $\geq R$, if links AS, BS and CS do not exist (capacity $= 0$).

The proofs of these two conditions are included in our technical report [10]. Necessary conditions NC1 to NC5, together with the algorithm we are presenting in this section prove that:

> Agreement capacity of a four-node network is the supremum over all throughputs R that satisfy necessary conditions NC1, NC2, NC3, NC4 and NC5.

It is worth pointing out that, according to NC5, the existence of the uplinks (links AS, BS, CS) changes the capacity dramatically: even if there is only one non-zero uplink and it has very low capacity $\epsilon \to 0$, the agreement capacity is significantly higher than the one of the same network with the ϵ uplink removed. For example, in the four-node network in Figure 1(b), even if links AS and BS are removed and link CS has ϵ capacity, the agreement capacity is 2. But once link CS is also removed, the capacity becomes 1.



(a) Labels denote some link capacities.

(b) Every directed link has capacity 1. Source node S sends packets $a, b, a \oplus b$ to A, B and C, respectively.

Fig. 1. Four-node network

The proposed algorithms achieve throughput arbitrarily close to R bits/unit time, for any R that satisfies NC1 to NC5. The algorithm for the *complete* four-node networks with all point-to-point links having **non-zero** capacity (although capacity of some links may be arbitrarily close to 0) is slightly different from the ones for incomplete four-node networks with < 3 uplinks, and is easier to describe. Due to space limitations, we only present the algorithm for the complete networks here. Details of the other algorithms can be found in [10].

Figure 1(a) shows a complete four-node network. The labels near the various links denote the link capacities. With this notation, condition NC1 implies, for instance, that $l + m \geq R$ and $t + m \geq R$; and NC2 implies that $t + u \geq R$.

The sequence of information bits are divided into generations of Rc bits (the choice of c will be elaborated later), and agreement are achieved one generation after another using the proposed algorithm. The proposed BA algorithm for complete four-node networks has four modes of operation, numbered I, II, III, IV. As seen later, repeated (and pipelined) execution of our algorithm can be used to achieve throughput approaching the capacity. At time 0 (the first generation), the network starts in mode I. The mode number may change as the algorithm is executed repeatedly over time, but it never decreases.[2]

In each mode, starting with the error detecting stage, the information sent by S is coded and scheduled such that either misbehavior by any node is detected, or the fault-free peers correctly agree without detecting the misbehavior. In the event that a misbehavior is detected, the full broadcast stage is performed. After this, each fault-free peer is able to narrow down the failure to a subset that contains one or two of the other nodes. The union of these subsets at the fault-free peers satisfies one of the following three properties: (i) the union contains two peers, or (ii) the union contains one peer and the source node S, or (iii) the union contains just one node. In these three cases, the fault-free peers change the mode for the following generations, respectively, modes II, III and IV (using a new error detection code).

[2]If the algorithm is modified to allow for the possibility of node repair, the mode number may indeed decrease. We ignore node repair in this paper. So at most one node fails for the entire time duration.

A. Operation in Mode I

This algorithm is motivated by Reed-Solomon codes and the prior work on network coding [8]. In particular, for a suitable value of parameter c (as elaborated below), each "packet" sent by the algorithm consists of 1 symbol from Galois field GF(2^c). One execution of the algorithm, which takes multiple rounds, allows the nodes to agree on R symbols from GF(2^c) that are sent by S.

The algorithm executes in multiple rounds, with the duration of each round being approximately c time units (as elaborated in Section IV-E). Note that in c time units, a link with capacity z bits/unit time can carry z symbols (or packets) from GF(2^c). Computation is assumed to require 0 time.[3] As an exception, an "extended round" requires longer duration, but as we will elaborate later, such extended rounds occur at most twice over infinite time interval.

In each generation, the source S encodes R packets of data, each packet being a symbol from GF(2^c), into $k + l + m$[4] packets according to some pre-defined linear code: each coded packet is obtained as a linear combination of the R packets of data. Let us denote the R data packets as the data vector

$$\tilde{x} \ = \ [x_1, x_2, \ldots, x_R]$$

and the $k + l + m$ coded packets as $y_1, y_2, \ldots, y_{k+l+m}$.

For the correctness of the algorithm, these $k+l+m$ packets need to be computed such that any subset of R encoded packets constitutes **independent** linear combinations of the R data packets. As we know from the design of Reed-Solomon codes, if c is chosen large enough, this linear independence requirement can be satisfied. The weights or coefficients used to compute the linear combinations is part of the algorithm specification, and is assumed to be correctly known to all nodes a priori. Due to the above independence property, **any** R of the $k + l + m$ symbols – if they are not tampered – can be used to (uniquely) solve for the R data packets.

Normally, mode I consists of 3 rounds and the operations in each round are described as follows

- Round 1: node S transmits k symbols $y_1, \cdots, y_k$ to node A, l symbols $y_{k+1}, \cdots, y_{k+l}$ to B, and $y_{k+l+1}, \cdots, y_{k+l+m}$ to node C, on links SA, SB and SC, respectively.
- Round 2: Each peer node forwards as many distinct packets received from node S as possible to the other two peers, in increasing order of their index. For instance, B forwards the $min(l, t)$ packets with the smallest index to C, that is, $y_{k+1}, \cdots, y_{k+min(l,t)}$.
- Round 3: Each fault-free peer checks the consistency of packets received from the other three nodes in rounds 1 and 2 and broadcasts to the remaining 3 nodes a 1-bit notification indicating whether the received packets are consistent or not.

[3] As seen later, we use the agreement algorithm in a pipelined manner. Computation delay can be incorporated by adding pipeline stages for computation, in addition to communication stages.

[4] Please refer to Figure 1(a) for the notation for link capacities.

If none of the notifications indicates a failure detected, then each fault-free peer agrees on the unique solution of the received packets, and the execution of the current instance of the algorithm is completed. However, if failure detection is indicated by any peer, then an "extended round 3" is added to the execution, as elaborated soon.

In round 3, consistency of the received packets is checked by finding the solution for **each** subset of R packets from among the packets received from the other three nodes in rounds 1 and 2. If the solutions to the various subsets are not unique, the packets are *inconsistent* and misbehavior of the faulty node is detected. This checking operation may seem to be quite computational expensive at first sight. However, since the code is known in advance, it can be done simply by checking whether the received packets consist a valid codeword. This can be achieved with complexity equals to the total size of the received packets, which is $O(Rc)$.

The 1-bit notifications are broadcast reliably using a traditional Byzantine agreement algorithm. For example, the one proposed by Pease, Shostak and Lamport [2]. Since at most one node is faulty, using this traditional algorithm, all fault-free nodes obtain identical 1-bit notifications from all the peers.

Figure 1(b) shows a simple example wherein node S has unit capacity outgoing links (that is, $k = l = m = 1$). In this illustration, as shown in the figure, for some values a and b, $x_1 = a$, $x_2 = b$, and $y_1 = x_1 = a$, $y_2 = x_2 = b$, and $y_3 = x_1 \oplus x_2 = a \oplus b$.

The following theorem states the correctness of the coding scheme we used in mode I. Its proof is in Appendix A.

Theorem 1: In mode I, misbehavior by a faulty node will either be detected by at least one fault-free peer, or all the fault-free peers will reach agreement correctly.

Extended Round 3 (full broadcast stage): As seen in round 3, an extended round is added subsequent to a failure detection. As seen below, the broadcast stage is quite expensive, but it is performed at most twice over time interval $[0, \infty)$. In round 3, the fault-free peers learn that some node has behaved incorrectly. The purpose of the *extended* round is to allow the nodes to narrow down the failure to a subset of the nodes. For this purpose, during the broadcast stage, every node (including the source) broadcasts all the packets it has sent to other nodes, or received from other nodes, during rounds 1 and 2 – as with the failure notifications in round 3, these broadcasts are also performed using the traditional Byzantine agreement algorithm. Since all links in our network have non-zero capacity, it is possible to use the traditional Byzantine agreement algorithm as desired here. However, since the capacity of some of the links may be very small, the time required for performing the broadcast stage in extended round 3 may be very large compared to the time required for performing the other rounds.

The fault-free nodes use the information received during the broadcast stage to narrow down the failure to a subset of nodes

(as explained below), and enter mode II, III, or IV, depending on the outcome of this assessment.

Each node forms a *diagnosis graph* after the broadcast stage in *extended round 3*, as elaborated next. The diagnosis graph contains four vertices S, A, B and C, corresponding to the four nodes in our network. there is an *undirected* edge between each pair of vertices, with each edge being labeled as g at time 0 (with g denoting "good"). The labels may change to f during extended round 3. Once a label changes to f (denoting "faulty"), it is never changed back to g. Without loss of generality, consider the edge between vertices X and Y in the diagnosis graph. The label for this edge may be set to f in two ways:

- For each packet transmitted in rounds 1 and 2 (sent by any node), each fault-free node will compare the claims by nodes X and Y about packets sent and received on links XY and YX. If the two claims mismatch, then the label for edge XY in the diagnosis graph is set to f.

- If node X is a peer and claims to have detected a misbehavior in round 3, but the packets it claims to have received in rounds 1 and 2 are inconsistent with this claim, then edge XY in diagnosis graph is set to f. In this case, all edges associated with X are set to f.

Similar actions are taken for each of the 6 undirected edges in the diagnosis graph. An example diagnosis graph thus obtained is illustrated in Figure 2(a). Due to the use of Byzantine agreement for the broadcast stage, all nodes will form identical diagnosis graphs. The notions of diagnosis and conflict graphs here are borrowed from past literature on *system-level diagnosis* [13], and continuous consensus [14], respectively.

It should not be difficult to see that the edge between vertices corresponding to fault-free nodes will remain g. For instance, if nodes A and B are fault-free, then edge AB in the diagnosis graph will remain g. So, if a link is marked as f, the faulty node must be one of the two nodes associated with this link. Additionally, we have the following theorem:

Theorem 2: When a failure is detected, at least one edge associated with the faulty node will be marked as f in the diagnosis graph after the broadcast stage.

The proof of this theorem can be found in Appendix B

According to Theorem 2, at least one edge will be marked as f. Since the faulty node must be one of the two nodes associated with an f-edge in the diagnosis graph, if there is only one f-edge, we can narrow down the faulty node to be **one** of the two nodes corresponding to the two endpoints of that edge – the set of these two nodes is called the "fault set". If there is more than one f-edge, then they must share a vertex in common, and the node corresponding to the common vertex must be the faulty node (recall that edge between vertices for two fault-free nodes can never become f). For instance, the diagnosis graph in Figure 2(a) implies that node S must be faulty. Depending on the outcome of this "diagnosis" using the diagnosis graph, the system enters mode II, III, or IV for the subsequent generations of data. In particular, when the "fault set" is narrowed down to two peers, the system enters mode II; if the fault set contains a peer and node S, the system enters mode III; and if the fault set contains only one node (the faulty node is known exactly to all other nodes), the system enters mode IV.

How much additional time is required for the *extended round 3*? Since we assume that the capacity of all links is > 0, it follows that the broadcast stage in extended round 3 will require a finite amount of time, although the duration would be large when link capacities are small. In particular, the number of bits that needs to be transmitted on each link during extended round 3 is $O(Rc)$, and with non-zero capacity on each link, the time required would be $O(Rc)$ as well. As we will see later, the broadcast stage occurs only once for each mode change, and mode change occurs at most twice. Thus, as we will also see later, in a pipelined execution, the negative impact (on the throughput) of extended round 3 becomes negligible as time progresses.

B. Operation in Mode II

Mode II is entered when the fault set is narrowed down to two peers. Without loss of generality, assume that the faulty node is narrowed down to the set $\{$A,B$\}$, and node A is actually the faulty node. Recall that the fault set is $\{$A,B$\}$ when only edge AB in the diagnosis graph is f. This also implies that node C does not know the exact identity of the faulty node, but knows that node S is definitely fault-free. Fault-free node B knows that nodes C and S are both fault-free.

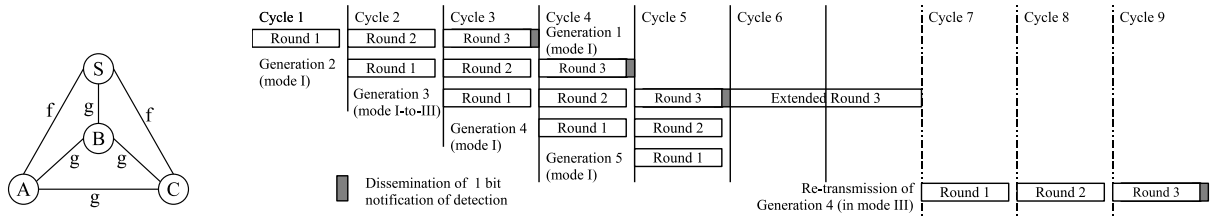
The operations in mode II is similar to the ones in mode I. Round 1 is the same as in mode I. Round 2 is the same as in mode I, except for that no packets are scheduled on links AB and BA. Since node B knows that nodes S and C are fault-free, and since it received at least R packets from S and C together (this follows from condition NC1), node B can use the packets received from S and C to recover the correct data.

In round 3, only node C checks for consistency of received packets and broadcasts a 1-bit notification as in mode I. If node C finds the packets consistent, then the corresponding solution must be identical to $\tilde{x}$ sent by node S (since node C receives at least R correct packets from S and the fault-free peer) – in this case, node C agrees on the unique solution. If there is no such unique solution, node C has detected an attack, although it does not yet know that node A is the faulty node. If and only if node C has detected a failure, *extended round 3* is performed as in mode I.

Similar to Theorem 2, we can show that after the broadcast stage, at least one more edge associated with the faulty node will be labeled f. Then all fault-free nodes learn the identity of the faulty node as the one associated with both f-edges. At this point, the system has transitioned to mode IV from mode II. The detailed discussion is in [10].

C. Operation in Mode III

Mode III is entered when the fault set is narrowed down to a set containing node S and one peer. Without loss of generality,



(a) An example of the diagnosis graph

(b) Example of pipelining: In generation 3, node A detects a mismatch and the system enters mode III. Generations 4 and 5 are then dropped and retransmitted after entering mode III.

Fig. 2. Examples of diagnosis graph and pipelining of the complete four-node network

assume that the location of the faulty node is narrowed down to the set {S,B}. Recall that the fault set is {S,B} when only edge SB in the diagnosis graph is f. In this case, nodes A and C know that they are both fault-free.

In mode III, the schedule in rounds 1 and 2 is the similar to that in mode I, with the only difference being that no transmissions are scheduled on links SB, BA, and BC. The schedule in round 3 needs to modified slightly.

As per NC1 and NC2, node B receives $\geq R$ coded packets from nodes A and C in round 2. In round 3, if node B is actually fault-free, it first tries to find the unique solution of the received packets. If the unique solution cannot be found, it detects an attack. Otherwise, node B computes packets $y_{k+1}, \ldots, y_{k+l}$ from the unique solution. Then, node B sends its own y-packets to nodes A and C in the same way as in round 2 of mode I. Then nodes A and C try to find the unique solution for the packets they have received, and appropriate 1-bit notifications are reliably broadcast at the end of round 3. Similar to theorem 1, it can be shown that nodes A and C will either reach an agreement correctly, or detect an attack. Moreover, in the case that node B is fault-free (which implies the sender S is faulty), it will also agree with node A and C correctly if no failure is detected. If a failure is indicated, extended round 3 is performed similar to mode I. A slightly different algorithm that does not "mix" packets in mode III is discussed in our technical report [10].

Since the fault set is {S,B}, either node S or node B must have tampered some packets. Similar to mode II, in this case as well, after the broadcast stage, one more edge in the diagnosis graph will become f. The common vertex to the two f-edges in the diagnosis graph will indicate the faulty node (also similar to mode II), and the system will transition to mode IV.

D. Operation in Mode IV

When the network is in mode IV, the identity of the faulty node is correctly known to all other nodes. In the event that node S is known to be faulty, the fault-free peers can decide to agree on some default value, and terminate the algorithm. In the event that a peer is known to be faulty, the fault-free nodes ignore all packets received from the faulty peer. In this case, in rounds 1 and 2, the schedule remains the same as in rounds 1 and 2 of mode I. Each fault-free peer can recover $\tilde{x}$ using the packets received from the fault-free source and the other fault-free peer (by condition NC1).

E. Throughput Analysis

With the exception of the extended round 3, each round in our algorithm uses identical time, which is slightly longer than c time units (why the round length is slightly longer than c will be clear soon). Observe that with the exception of the dissemination of 1-bit notification and the broadcast stage (in extended round 3), the usage of each link is within the link capacity in each mode. In particular, a link with capacity z carries at most z **data** packets combined over all rounds in each mode (ignoring the extended rounds). In achieving rate R, it will be necessary to have multiple "generations" of packets in the network, with the algorithm operating in a pipelined manner (one round per pipeline stage). Agreement algorithm for one new generation of data of size Rc bits (or R symbols from GF(2^c)) starts per "clock cycle", as shown in Figure 2(b). Each generation consists of three rounds and the packets are scheduled according to the schedules we discussed above. By the end of the round 3 of every generation, the peers exchange 1-bit notifications indicating whether they detected an attack. Subsequently, if necessary, an extended round 3 is performed. By the end of the extended round 3, if any, all nodes decide on the same new mode.

Figure 2(b) shows an example execution of the pipelining. The system starts in mode I and enters mode III after an failure detected in round 3 of generation 3. Note that, at the time the system enters mode III, two generations of packets are in the system using the old schedule for mode I (in this example: generations 4 and 5). To allow a transition to the new schedule in mode III, the packets belonging to generations 4 and 5 are dropped, and retransmitted using the algorithm/schedule for mode III. Thus, agreement for the dropped generations is re-initiated in subsequent clock cycles.

Recall that there are at most two mode transitions throughout the execution of the algorithm (mode I to II to IV, or mode I to III to IV), thus requiring at most two extended rounds. The time overhead of an extended round 3 is $O(Rc)$ – during the extended round, no new generations make progress. Since the extended round occurs at most twice, the average overhead *per cycle* of extended round 3 decreases to zero as time increases. In particular, the overhead per cycle becomes $O(\frac{1}{Rc})$ after R^2c^2 cycles. The dissemination of 1-bit notifications (in round 3) requires a fixed number of bits per link, independent of R and c. Thus, the total time overhead for this operation is $O(1)$. Thus, we can make the duration of each round to be equal to $c + O(1)$. Since a new generation of Rc bits worth

of information is initiated in each round, it follows that the agreement throughput is $R-O(1/c)$. Thus, by using a suitably large c, the throughput can be made arbitrarily close to R.

Thus, we have shown that given conditions NC1 and NC2 are satisfied, our algorithm achieves agreement throughput approaching R bits/unit time in a complete four-node network.

V. Symmetric Networks

In this section, we present an algorithm that achieves the agreement capacity for symmetric networks with arbitrary size $n \geq 4$ and up to $t < n/3$ faulty nodes, in which every link has capacity C bits/unit time. Thus, each pair of nodes is connected by two directed point-to-point links, each with capacity C. With the traditional agreement algorithms, such as [2], to achieve agreement on 1 packet of data, the source sends the whole packet on every link pointing to the peers. As a result, the agreement throughput achieved with these algorithms is upper bounded by the link capacity of each out-going link from the source, which in this case is C bits/unit time. By comparison, using our approach, it is possible to achieve a substantially higher throughput in the symmetric network. In particular, it can be shown that:

Given per-link capacity C, agreement capacity of a symmetric network of size n and up to $t < n/3$ failures, is $(n-t-1)C$.

The algorithm in this section also proceeds in generation and rounds as the one in section IV for four-node networks. Since all links have the same capacity, we will just consider that at most 1 packet can be sent on each link per generation, excluding the broadcast stage.

A. When no failure is yet detected

Let us label the peers as nodes $1, 2, \ldots, n-1$. The algorithm proceeds in rounds in a similar way as in the four-node networks:

- Round 1: The source node first encodes $n-t-1$ data packets into $n-1$ coded packets, namely $y_1, \ldots, y_{n-1}$, such that any subset of $n-t-1$ coded packets constitutes independent linear combination of the data packets. Then the source sends y_i to peer i.
- Round 2: After receiving packet y_i, peer i forwards it to all other peers.
- Round 3: Every fault-free peer checks the consistency of the packets, and broadcasts a 1-bit notification, in a same way as in the algorithm for four-node networks.

Similar to Theorem 1, we can prove that whenever some faulty nodes misbehave, the fault-free peers either reach agreement correctly, or at least one of them detects the failure. When a failure is detected, the extended round 3 is performed and the diagnosis graph is updated in the same way as in the four-node networks. If any node is associated with more than t f-edges, it must be faulty and is removed from the network.

B. After some failures have been detected

Let us say that two nodes accuse/trust each other if they are connected with an f/g edge in the diagnosis graph. Let us denote the set of peers trusted by the source as T, and the set of peers not trusted by the source as Q.

Observe that, in the case the source is fault-free, there must be a set $P \subset T$ of $n-t-1$ peers that all trust each other and all trust the source. So at the beginning of a generation, all fault-free peers try to find such a set P in the diagnosis graph. If such a set P does not exist, then it is with certainty that the source is faulty, and the fault-free peers will decide on some default value and terminate the algorithm. Otherwise, the algorithm proceeds as follows:

- Round 1: Source node sends 1 coded packet to each of the peers in T.
- Round 2: Every peer $j \in T$ (trusted by source) sends the packet y_j to all other peers in T that it trusts.
- Round 3: For every peer $i \in Q$ (accused by source), the algorithm selects a node $p_i \in P$ that node i trusts. Peer p_i mixes the packets it has received in rounds 1 and 2 into a new coded packet z_i. Then, every peer $j \in T$ sends y_j to its trusted peers in Q, with the exception that if $j = p_i$, j sends to node i packet z_i instead of y_j, for each $i \in Q$. After receiving these packets, each node $i \in Q$ then sends z_i to all peers it trusts.

The node p_i must exist, otherwise node i is accused by all $n-t-1 > t$ nodes in P, which implies that node i is faulty and must have been removed. For the correctness of the algorithm, any subset of $n-t-1$ of the union of y and z-packets must be linear independent. This is achievable because every p_i can first solve the data packets using the $n-t-1$ y-packets it has exchanged with nodes in P during round 2, then generate z_i in the same way as y_i. Now it is not hard to see that every pair of fault-free nodes share at least $n-t-1$ coded packets, so the fault-free peers will either agree correctly or detect a failure.

The rest of the algorithm (broadcast stage and updating diagnosis graph) is the same as before. Through pipelining, it achieves throughput arbitrarily close to $(n-t-1)C$. On the other hand, according to NC1 - NC3, the agreement capacity of the symmetric network is upper bounded by $(n-t-1)C$. Thus, we can conclude that the capacity of the symmetric network is $(n-t-1)C$.

VI. Related Work

A. Prior work on agreement or consensus

There has been significant research on agreement in presence of *Byzantine* or *crash* failures, theory (e.g., [2], [15], [16]) and practice (e.g., [17], [18]) both. Perhaps closest to our context is the work on *continuous consensus* [19], [20] and *multi-Paxos* [21], [17] that considers agreement on a long sequence of values. For our analysis of throughput as well, we will consider such a long sequence of values. However, to the best of our knowledge, the past work on multi-Paxos

and continuous consensus has not addressed the problem of optimizing throughput of agreement while considering the *capacities of the network links*. Some of the past work has analyzed *number of bits* needed to achieve agreement. While this is related to the notion of capacity or throughput, such prior work disregards the capacity of the links over which the data is being carried. Link capacity constraints intimately affect capacity of agreement. Past work has explored the use of error-correcting codes for asynchronous consensus (e.g., [22]). Our algorithms also use error detecting codes, but somewhat differently.

B. Prior work on multicast using network coding:

While the early work on fault tolerance typically relied on replication [23], [24] or source coding [25] as mechanisms for tolerating packet tampering, *network coding* has been recently used with significant success as a mechanism for tolerating attacks or failures. In traditional routing protocols, a node serving as a router, simply forwards packets on their way to a destination. With network coding, a node may "mix" (or *code*) packets from different neighbors [6], and forward the coded packets. This approach has been demonstrated to improve throughput, being of particular benefit in *multicast* scenarios [6], [26]. The problem of *multicast* is related to *agreement*. There has been much research on multicast with network coding in presence of a Byzantine attacker (e.g., [27], [7], [8], [28], [9]). The significant difference between Byzantine agreement and multicasting is that the multicast problem formulation assumes that the source of the data is always fault-free.

VII. Conclusion

In this paper, we studied the *capacity* of Byzantine agreement under the constraints of a finite capacity of point-to-point links in the network. We identified necessary conditions of the achievable throughputs in general networks. Then we introduced a structure for the capacity achieving agreement algorithm in general networks. In addition, we presented capacity achieving algorithms for two classes of networks: four-node networks with arbitrary link capacity with at most 1 faulty node, and symmetric networks with arbitrary size n and at most $t < n/3$ faulty nodes.

Acknowledgment

This research is supported in part by 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. We thank Pramod Viswanath and Jennifer Welch for their feedback.

References

[1] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *JOURNAL OF THE ACM*, 1980.

[2] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. on Programming Languages and Systems*, 1982.

[3] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, pp. 398–461, November 2002. [Online]. Available: http://doi.acm.org/10.1145/571637.571640

[4] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," *ACM Trans. Comput. Syst.*, vol. 27, pp. 7:1–7:39, January 2010. [Online]. Available: http://doi.acm.org/10.1145/1658357.1658358

[5] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.

[6] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *Information Theory, IEEE Transactions on*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[7] N. Cai and R. W. Yeung, "Network error correction, part ii: Lower bounds," *Communications in Information and Systems*, 2006.

[8] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks using randomized network coding," 2004.

[9] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient network coding in the presence of byzantine adversaries," in *INFOCOM'07*, 2007.

[10] G. Liang and N. Vaidya, "Capacity of byzantine agreement: Complete characterization of the four node network," *Technical Report, CSL, UIUC*, April 2010.

[11] O. Kosut and L. Tong, "Nonlinear network coding is necessary to combat general byzantine attacks," in *47th Annual Allerton Conference on Communication, Control, and Computing*, October 2009.

[12] G. Liang, R. Agarwal, and N. Vaidya, "Secure capacity of wireless broadcast networks," *Technical Report, CSL, UIUC*, September 2009.

[13] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Trans. Electr. Comput.*, vol. EC-16, no. 6, pp. 848–854, Dec. 1967.

[14] T. Mizrahi and Y. Moses, "Continuous consensus with ambiguous failures," *Distributed Computing and Networking (Lecture Notes in Computer Science)*, vol. 4904/2008, pp. 73–85, 2008.

[15] N. A. Lynch, *Distributed algorithms*. Morgan Kaufmann, 1995.

[16] H. Attiya and J. Welch, *Distributed Computing*. McGraw-Hill, 1998.

[17] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: an engineering perspective," in *PODC '07*. New York, NY, USA: ACM, 2007, pp. 398–407.

[18] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, 2002.

[19] T. Mizrahi and Y. Moses, "Continuous consensus with failures and recoveries," in *DISC '08*, 2008.

[20] ——, "Continuous consensus via common knowledge," in *TARK'05*, 2005.

[21] L. Lamport and K. Marzullo, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, pp. 133–169, 1998.

[22] R. Friedman, A. Mostefaoui, S. Rajsbaum, and M. Raynal, "Distributed agreement and its relation with error-correcting codes," in *Proc. 16th Int. Conf. Distributed Computing*, 2002.

[23] E. C. Cooper, "Replicated distributed programs," in *SOSP'85*, 1985.

[24] M. Chereque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron, "Active replication in delta-4," in *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, Jul 1992, pp. 28–37.

[25] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, 1989.

[26] R. Koetter and M. Medard, "An algebraic approach to network coding," in *ISIT'01*, 2001.

[27] R. W. Yeung and N. Cai, "Network error correction, part i: Basic concepts and upper bounds," *Communications in Information and Systems*, 2006.

[28] S. Kim, T. Ho, M. Effros, and S. Avestimehr, "New results on network error correction: capacities and upper bounds," in *ITA'10*, 2010.

Appendix A
Proof of Theorem 1

Node S is said to misbehave only if it sends packets to A, B and C that are inconsistent – that is, all of them are not appropriate linear combinations of an identical data vector $\tilde{x}$.

A peer node is said to misbehave if it forwards tampered (or incorrect) packets to the other peers.

Faulty peer: Without loss of generality, suppose that peer A is faulty. Due to condition NC1, and the manner in which packets are forwarded in rounds 1 and 2, each fault-free peer (that is, B or C) receives R untampered packets, either directly from S or via the other fault-free peer. The solution of these R packets must be the correct R symbols from node S. Thus, any tampered packets sent by node A to B or C cannot cause the fault-free peer to agree on any data other than the correct data $\tilde{x}$. It then follows that, either the fault-free peers agree on the correct data, or detect the faulty behavior by node A (although they will not yet be able to determine that the faulty node is specifically node A) .

Faulty sender S: Now, let us consider the case when node S is faulty. Thus, all the peers are fault-free. For convenience, with an abuse of notation, we will denote the capacity of link XY as XY (instead of using the notation in Figure 1(a)). From condition NC2, it follows that $BA+CA \geq R$, $AB+CB \geq R$ and $AC+BC \geq R$. This implies that, $(AB+BA)+(BC+CB)+(AC+CA) \geq 3R$. Therefore, at least one of the terms $(AB+BA)$, $(BC+CB)$, and $(AC+CA)$ must exceed R. Without loss of generality, suppose that $AB+BA \geq R$.

Now, let us consider the number of packets nodes A and B exchange in round 2 on links AB and BA together. Observe that A sends $min\{SA, AB\}$ packets to B on AB, and B sends $min\{SB, BA\}$ to A on link BA. So the number of packets they exchange on links AB and BA together is

$$\begin{aligned} & min\{SA, AB\} + min\{SB, BA\} \\ = & \, min\{SA+SB, SA+BA, AB+SB, AB+BA\} \quad (1) \\ \geq & \, min\{R, AB+BA\} \quad (2) \end{aligned}$$

The reason for the $\geq$ in Equation 2 is that each of the first three terms on the right hand side of Equation 1, namely $SA+SB$, $SA+BA$, $AB+SB$, $\geq R$, as per condition NC1. $AB+BA \geq R$ and 2 together imply that after round 2, nodes A and B share at least R packets. That is, among the packets A and B have received, there are R identical packets. Thus, A and B will not agree on different data symbols (since the agreed data must satisfy linear equations corresponding to all received packets), even though S may be misbehaving.

Thus, either at least one of A and B will detect misbehavior by node S, or all the packets they have received will be consistent with an identical data vector (of R symbols). In the former case, the misbehavior by S is already detected (although the fault-free peers may not yet know that S is the faulty node). In the latter case, neither A nor B detects the misbehavior. In this case, we will now consider what happens at node C. In particular, there are three possibilities:

- $AC+CA \geq R$: Similar to the above argument for nodes A and B, we can argue that nodes A and C will have at least R packets in common, and therefore, they will not agree on two different data vectors. This, combined with the fact that A and B will also not agree on two different data vectors, implies that if none of the three fault-free peers detects a misbehavior, then they will agree on an identical data vector.
- $BC+CB \geq R$: This case is similar to the previous case.
- $AC+CA < R$ and $BC+CB < R$: In this case, similar to Equation 2, we can show that:

  $min\{SA, AC\} + min\{SC, CA\} \geq min\{R, AC+CA\}$

  $min\{SB, BC\} + min\{SC, CB\} \geq min\{R, BC+CB\}$.

  This implies that these four links are all "saturated" (that is, the number of packets sent on each of these links in round 2 is equal to the link capacity). Since $AC+BC \geq R$ (by condition NC2), it follows that $AC+CA+BC+CB \geq R$, and that node C has at least R packets in common with the union of packets available to nodes A and B. Since nodes A and B have not detected a misbehavior, these R packets must all be consistent with the solution obtained at nodes A and B both. Thus, node C cannot possibly agree on a data vector that is different from that agreed upon by A and B. Thus, it follows that, either at least one of the peers will detect the misbehavior by node S, or they will all agree.

APPENDIX B
PROOF OF THEOREM 2

We will consider the cases when S is faulty and a peer is faulty separately.

A peer is faulty: Without loss of generality, suppose that node A is faulty. It misbehaves either by (i) raising a false alarm (implicitly accusing another node of misbehavior) or (ii) by sending some tampered packets to nodes B or C. Let us consider each case. (i) If node A broadcasts only correct packets in the broadcast stage, that will contradict with the false alarm, and edges BA, CA and SA in the diagnosis graph will all be labeled f. On the other hand, if node A broadcasts incorrect packets, inconsistent with packets actually received from node S/B/C, then edge SA/BA/CA will be labeled f (note that when A is faulty, S, B and C are fault-free). (ii) Now suppose that node A has misbehaved by sending incorrect packets to another peer: In this case, if A broadcasts correct packets in the broadcast stage, the edge between the recipient of the tampered packets from A (in round 2) and node A will be marked f. Otherwise, edge SA will be marked f similar to case (i).

Node S is faulty: Node S can misbehave only by sending $k+l+m$ packets in round 1 such that all subsets of R packets do not have a unique solution. During the broadcast stage, if S broadcasts packets such that subsets of R packets do not have a unique solution, then it is clear that node S is misbehaving, and edges SA, SB and SC in the diagnosis graph are all marked f. On the other hand, if the packets broadcast by S in the broadcast stage all have a unique solution, then the packets received by at least one peer in round 1 will differ from packets sent by S in the broadcast stage. Thus, the edge between that peer and S in the diagnosis graph will be marked f.