Capacity of Byzantine Agreement

(Preliminary Draft – Work in Progress)

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

Technical Report

January 15, 2010

Caveat: This report represents work-in-progress, and provides some of our early results on the topic. A more complete version of this report, with additional results not presented in this draft, and improved description of the algorithms and analysis, will be released later this year. This draft version contains known problems with clarity/readability and the descriptions are incomplete/imprecise in some places. The draft is being released nevertheless to seek early feedback from other researchers.

1 Introduction

In this report, we consider the problem of maximizing the throughput of Byzantine agreement. Byzantine agreement is a classical problem in distributed computing, with initial solutions presented in the seminal work of Pease, Shostak and Lamport [33, 22]. There has also been more recent work on designing multicast algorithms that can survive Byzantine attacks (e.g., [13]). Section 2 summarizes the past work related to this report.

Many variations on the Byzantine *agreement* problem have been introduced in the past, with some of the variations also called *consensus*. We will use the original definition of Byzantine agreement in our discussion. In particular, we consider a network with one node designated as the *sender* or *source*, and other nodes designated as the *peers*. The goal here

is for all the fault-free nodes to "agree on" the value being sent by the sender. 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 value greed on by the peers must be identical to sender's value.

Once a node agrees on a certain value, it cannot change its decision. It is customary to also include a *termination* condition [1], which states that the peers must eventually agree on a value. We will revisit the notion of termination below.

Our goal in this work is to design algorithms that can improve *throughput* of agreement. The notion of throughput here is similar to that used in the networking/communications literature on unicast or multicast traffic. We now define *agreement throughput* (or throughput of agreement) somewhat more formally.

When defining throughput, the "value" referred above in the definition of agreement is viewed as an infinite sequence of bits. 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 bit number i in the array, that agreed bit cannot be changed. In the discussion below, we assume that agreement algorithm begins execution at time 0. The system is assumed to be synchronous.

Throughput: For defining throughput, we consider the largest *prefix* of the array at all the peers that has been agreed upon. In particular, suppose that at time t, all the peers have agreed upon bits 0 through $b(t)$, and at least one peer has not yet agreed on bit number $b(t) - 1$, then the largest agreed prefix at time t has size $b(t)$. Agreement throughput T is defined as

$$T = \lim_{t \to \infty} \frac{b(t)}{t} \tag{1}$$

Although we assume that the information granularity is a bit, the definition can be modified trivially for symbols of larger size.

We do not explicitly define a termination condition. However, achieving non-zero throughput implicitly requires that agreement be eventually achieved on an infinite prefix of the information. In the above definition, we also do not restrict the delay in reaching the agreement. However, the definition may be modified to include additional constraints.

Capacity: 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 throughputs of agreement between the given sender and the peers.

Note that, in general, not all nodes in the network may be peers. That is, the network may include nodes that are neither sender nor peers. Such nodes may assist in achieving agreement.

The above definitions of throughput and capacity can be extended to other forms of agreement or consensus as well. For instance, *consensus* is often defined as agreeing on an identical value as a function of the values being proposed by all the peers (so, all the peers act as "senders"). In particular, if all the fault-free peers propose an identical value, then the agreed value should be this particular value. It should be easy to see that the above definition of throughput can be extended to this scenario as well. It should also be easy to see that the actual throughput for the different versions of consensus or agreement may be quite different.

The above definitions can be extended to the more general problem of *function computation*, where the agreed value may be an arbitrary function of the values proposed by the various nodes, and also to the notion of *approximate agreement* (in case of approximate agreement, one can trade-off throughput with "accuracy" of agreement).

While all of these issues are quite interesting, in this report, we consider the simplest instance of the agreement problem, as elaborated later.

2 Related Work

Research in three areas is most relevant to this report:

- Distributed fault-tolerant computing: There has been significant research on tolerating *Byzantine* failures, theory (e.g., [23, 29, 1]) and practice (e.g., [5, 3, 4, 19]) both. Perhaps closest to our context is the work on *continuous consensus* [32, 8, 9, 31] and *multi-Paxos* [21, 5] 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. In fact, [5] presents measurements of *throughput* achieved in a system that uses multi-Paxos. 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*.

  In general, most research on Byzantine agreement tends to focus on other metrics, such as the number of "rounds" of messages exchanged (as a measure of delay incurred), or the total number of messages, or number of bits that need to be exchanged to

achieve agreement. The last metric (number of bits) may seem related to the notion of capacity or throughput, however, such prior work disregards the capacity of the links over which the data is being carried. The link capacity constraints intimately affect capacity of agreement.

The work on the relationship between error-correcting codes and asynchronous consensus algorithms (e.g., [11]) is also related to our work. In particular, our agreement algorithm can be viewed as using a form of error correcting code.

- Multicast using network coding: While the early work on fault tolerance typically relied on replication [7, 6] or source coding [36] 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 received from different neighbors [25], and forward the coded packets (for instance, data in a forwarded packet may be obtained as bit-wise exclusive OR of the data in packets received from two neighbors). This approach has been demonstrated to improve throughput, being of particular benefit in *multicast* scenarios [25, 17, 10]. The problem of *multicast* is related to *agreement*. A similarity between this report and the work on multicasting with network coding is the use of capacity or throughput as the metric of interest. The significant difference between the two is that the multicast problem formulation assumes that the source of the data is always fault-free. As a simple but illuminating example of the impact of this difference, consider a source node S connected to two nodes A and B with a point-to-point link of capacity R each. In this three node network, it should be easy to see that multicast capacity of R is achievable; however, throughput of agreement in presence of a Byzantine node failure cannot exceed zero.

  There has been much research on multicast with network coding in presence of a Byzantine attacker (e.g., [38, 2, 13, 14]). Some of these solutions use carefully designed digital signatures [15, 39, 40, 24] or hash functions [20, 12], which allow intermediate nodes to verify the integrity of coded packets. Non-cryptographic solutions have also been proposed (e.g., [13, 14]), which exploit redundancy in the network such that the destination(s) can *correct* some packet tampering and recover correct information. However, most of the past networking coding literature on Byzantine attackers has considered a somewhat different Byzantine attacker model – in particular, it essentially assumes that an attacker is constrained only by the total rate at which it can tamper packets in the network, not by which links these packets belong to. The node-level fault model in our work assumes that a faulty or compromised node can only tamper packets on its outgoing links (but on these links, all packets may be tampered). There is some recent research on using this model to study capacity of unicast/multicast traffic [26, 27, 18, 16]. It can be shown that, even for unicast traffic, presence of

Byzantine node-level adversary, may require the use of non-linear coding strategies [28].

It should also be easy to see that the problem of Byzantine agreement is strictly harder than that of multicasting with a fault-free sender. The problem of multicast in presence of Byzantine node failures in general networks remains open at this time. In this report, we make progress on the problem of agreement by considering the simplest network topology that allows agreement to be achieved. It is hoped that the insights gained from this work will be useful in more general networks.

- System-level diagnosis: The work on system-level diagnosis considers a directed *diagnosis graph* (or a *test graph*), wherein each directed edge represents a *test*: in essence, when node X tests node Y, it may declare Y as faulty or fault-free, with a faulty tester providing potentially erroneous test outcomes. The seminal work of Preparata, Metze and Chien [34] identified conditions under which it is possible to correctly identify the faulty nodes in some networks. Much research has followed further developing the ideas introduced in [34], as summarized in [30]. The notion of *diagnosis graph* used in our algorithm is borrowed from the system-level diagnosis literature – it is also similar to the notion of *conflict graph* used in prior work on consensus [37].

3 Four Node Network

In this report, we primarily focus on a rather simple network consisting of just four nodes, with the goal of tolerating at most one Byzantine fault. It is known that tolerance of one Byzantine fault requires that the network contain at least four nodes [22] with node connectivity at least three.

In particular, consider a synchronous system consisting of four nodes, namely, S, A, B and C, with node S acting as the sender, and nodes A, B and C being the peers. At most one of these four nodes may be faulty.

We assume that each pair of nodes is connected by a pair of directed point-to-point links. Each point-to-point link has a certain *link capacity*, defined as the maximum rate at which information may be transmitted over that link.[1]

Figure 1 shows the four-node network. The labels near the various links denote the link capacities. Without loss of generality, we assume that $k \leq l \leq m$.

[1] In some instances, it is not necessary to have links in both directions between certain pairs of nodes. For brevity, we ignore this possibility here, and consider the case where links exist in both directions. In fact, we assume that a non-zero link capacity is available for all directed links.

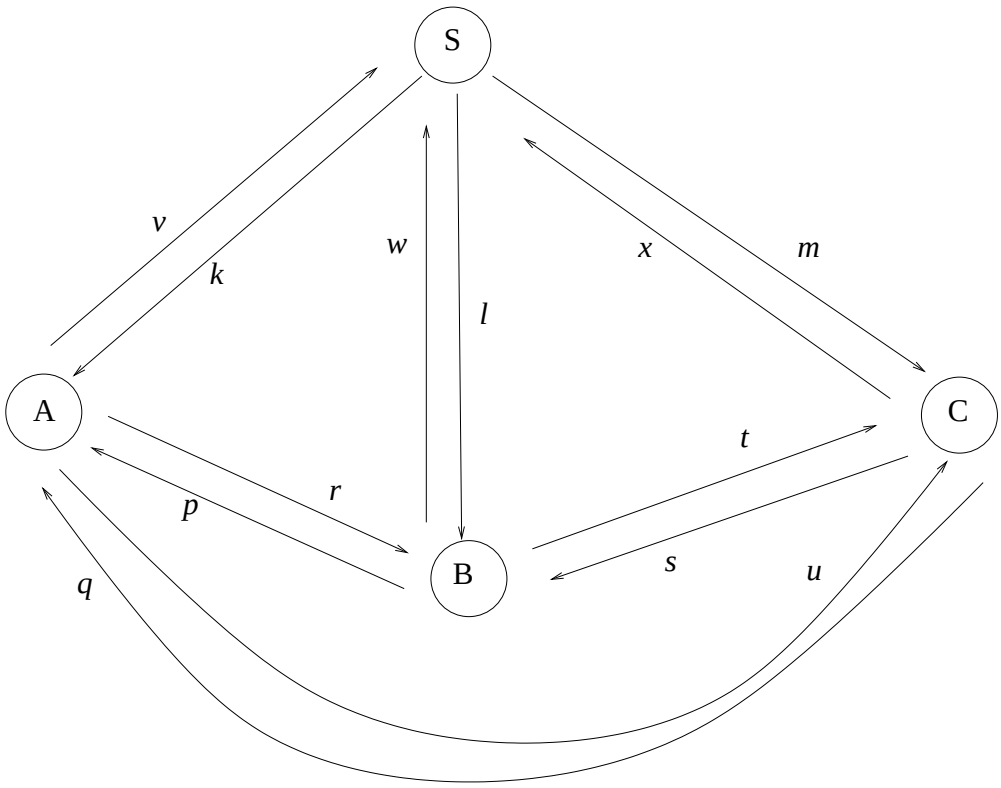

Figure 1: Four node network: Labels denote the link capacities. Without loss of generality, we assume that $k \leq l \leq m$.

4 Byzantine Agreement Algorithm

4.1 Link Capacity Constraints

We will present a Byzantine algorithm for the network in Figure 1. The algorithm can achieve throughput arbitrarily close to R bits/unit time, provided that the following sets of conditions are true.

- If one of the peers is removed from the network, then these conditions must be true for the min-cut from S to a remaining peer to be at least R:

$$k + l \geq R \tag{2}$$
$$l + m \geq R \tag{3}$$
$$m + k \geq R \tag{4}$$
$$p + k \geq R \tag{5}$$
$$q + k \geq R \tag{6}$$

$$r + l \geq R \tag{7}$$
$$s + l \geq R \tag{8}$$
$$t + m \geq R \tag{9}$$
$$u + m \geq R \tag{10}$$

The above conditions are *necessary* for agreement throughput of R to be achieved. When one of the peers is faulty, it may not communicate with the other nodes at all, while all the remaining nodes behave correctly. Thus, the agreement throughput cannot exceed the max-flow (or min-cut) from S to any peer, with one of the other peers removed from the network.

- These conditions must hold for the max-flow to one of the peers from the other two peers, with sender S removed, to be at least R.

$$p + q \geq R \tag{11}$$
$$r + s \geq R \tag{12}$$
$$t + u \geq R \tag{13}$$

The above conditions are *necessary* for throughput of R to be achieved. To see this, we consider two scenarios.

Scenario 1: Suppose that node A misbehaves essentially by pretending that links SA and AS are broken (thus, it will not transmit packets to S, and any packets it would otherwise have to sent to S are simply dropped). Thus, no communication takes place on link SA. Aside from this, node A behaves correctly. Since S is fault-free, nodes B and C must agree on the information sent by S.

Scenario 2: Now consider a scenario in which node S is faulty (so node A is fault-free). Node S misbehaves only by pretending that links SA and AS are broken.

In essence, both scenarios are equivalent to the scenario in which links SA and AS are broken. Thus, from the perspective of B and C, these two scenarios are not distinguishable. Thus, in both scenarios, nodes B and C agree on identical information.

In scenario 2, node A is, in fact, fault free. So it must also agree on the information agreed on by nodes B and C. The only way for node A to learn this information is to receive it from nodes B and C. Thus, the agreement throughput is upper bounded by the min-cut from {B,C} to A with node S removed.

Similar arguments can be made with nodes B or C replacing node A in the above scenarios.

- If the source and a peer is removed, for the "undirected min-cut" between the remaining two peers (with the links being treated as undirected) to be R, these conditions must be true:

$$p + r \geq R \tag{14}$$
$$s + t \geq R \tag{15}$$
$$q + u \geq R \tag{16}$$

  All the conditions in the third set of constraints are not necessary for achieving agreement. This can be demonstrated by constructing networks wherein agreement can be reached even if some of these conditions are not true. These conditions, together with the other constraints, are sufficient to achieve agreement at rates approaching R, as demonstrated by the existence of an agreement algorithm under these conditions.

- We also impose the constraint that the capacity of each directed link is more than 0. This allows each node to transmit a fixed number of bits to another node within a finite interval of time. In general, non-zero capacity is not necessary for all links in the network. However, the algorithm presented in this report requires this condition to be true.

The proposed agreement algorithm can achieve agreement throughput approaching R bits/unit time, where the unit of time is defined to be of a suitable size. Note that if the unit of time is increased (say, from 100 ms to 1 second), then the numerical value of R will also increase proportionately.

To simplify the discussion below, unless otherwise specified, we will normalize all quantities by R. Thus, the goal of the algorithm is to achieve normalized throughput approaching 1. Similarly, link capacity k for link SA should be viewed as being normalized, with the actual capacity in bits/unit time being kR. We will also normalize the packet sizes by R. For instance, when we say that the (normalized) packet size is $1 - m$, the actual packet size in bits is $(1 - m)R$ – note that, m here is also normalized with R.

Note that in inequalities 2 through 13, R should be replaced 1 to obtain the inequalities for normalized quantities.

Observe that if any of the inequalities above holds, then the inequality also holds if we replace a link rate, such as k by $minimum(1, k)$. Thus, in the algorithm design, we will assume that each link rate is at most 1.

The agreement algorithm has three different modes of operation. We will number these modes as I, II, III, IV. As seen later, repeated (and pipelined) execution of our algorithm below can be used to achieve the desired throughput. The network starts in mode I,

and may change modes over time as the algorithm is executed multiple times. The mode number does not decrease over time[2]).

- Mode I: *Fault-free*: The network starts operation in this mode.
- Mode II: Failure known to be within a set containing two peers.
- Mode III: Failure known to be within a set containing a peer and the source.
- Mode IV: Source node known to be faulty. and *source* interchangeably.

The 1 unit of data is divided into 6 parts or "packets", named D through H. For the packet sizes below to be meaningful, we assume that $k \leq l \leq m \leq 1$.

- D of size $k + l - 1$,
- E of size $m - l$,
- F of size $1 - m$,
- G of size $m - l$,
- H of size $1 - m$, and
- I of size $l - k$.

The packet sizes are normalized by R. Thus, normalized packet size $(l - k)$ really means $(l - k)R$ bits. We assume that with appropriate scaling factors, the normalized packet sizes can all be turned into integers (this is true if k, l, m are rational numbers). Note that the normalized packet sizes add up to 1.

In the algorithm presented below, packets G and I of size $m-l$ and $l-k$, respectively, are treated identically, and as such, could be replaced by a single packet of size $m - k$. The above notation is residual from a prior version of the agreement algorithm presented here.

To aid the discussion, Figure 2 shows some of the link capacity constraints listed above. The figure also shows some packets transmitted on various links; the significance of these will be clearer below.

[2]The algorithm can be modified somewhat to allow for the possibility of node repair. In this case, the mode number may indeed decrease. In this report, we do not consider the possibility of repair, and assume that at most one node can fail over the infinite time duration.

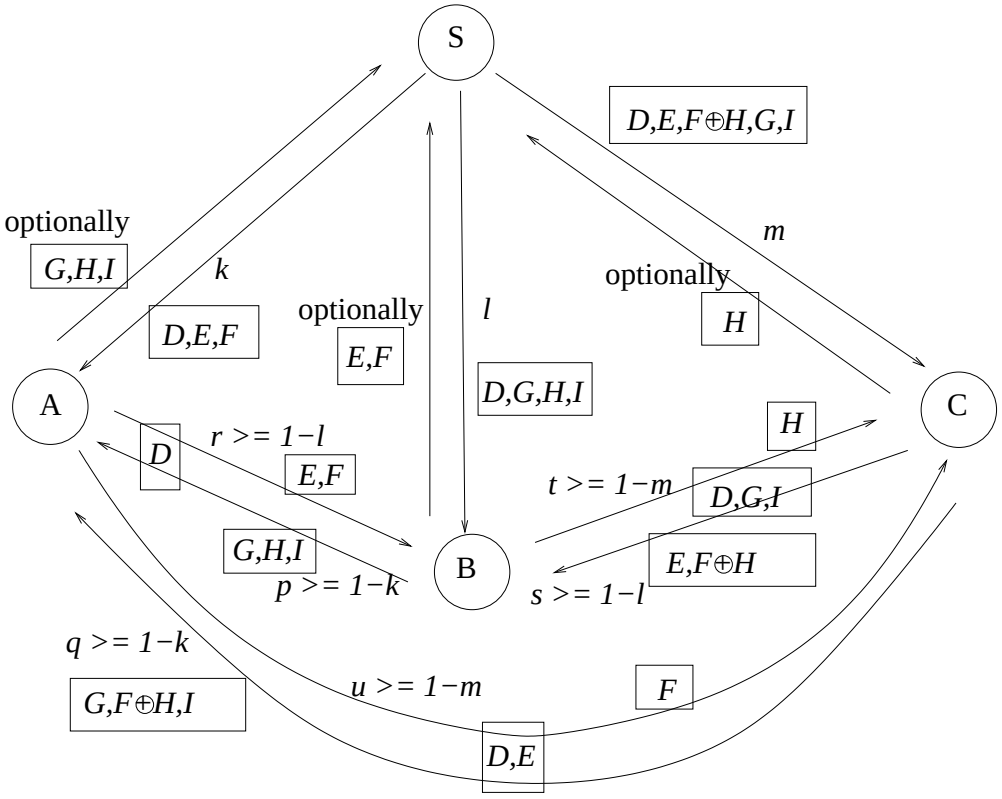

Figure 2: Four node network – Some link rate constraints shown

4.2 Operation in Modes I and II

The operation in modes I and II is substantially identical. Therefore, we discuss these two modes together. The network is initially in mode I, and remains in this mode until the detection of the presence of a node failure, or incorrect behavior by some node. The new mode subsequent to failure detection may be II, III, of IV, as elaborated later.

The Byzantine agreement algorithm proceeds in rounds. The duration of most rounds can be made approximately 1 unit time, as discussed later. Computation is assumed to require 0 time.[3]

Round 1: Sender S transmits as follows:

D, E, F to node A,

D, G, H, I to node B, and

[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.

$D, E, G, I, F \oplus H$ to node C.

These transmissions are shown in Figure 2.
The number of bits sent on links from S to the peers are as follows:
link SA $= k$, link SB $= l$, and link SC $= m$.

Round 2: The peers send information to each other according to the following schedule. The schedule is illustrated in Figure 2

- Links AB and BA: Schedule packets E, F on link AB, and packets G, H, I on link BA. Size of E, F together is $1-l$, and size of G, H, I is $1-k$. Thus, we are within capacity r and p of links AB and BA, respectively (by inequalities 7 and 5, respectively). Recall that in inequalities 2 through 13, R needs to be replaced by 1, to obtain inequalities for normalized rates.

  Recall from inequality 14 that $p + r \geq 1$, and that the total size of D, E, F, G, H, I is 1. Thus there is capacity left on links AB and BA together to transmit packet D. Schedule part of D on AB, and remaining part of D on BA, such that all of D is scheduled, and the capacity of links AB and BA is not exceeded. Recall that both A and B receive D from S.

  Observe that now A and B exchange all packets with each other on links AB and BA together.

- Links BC and CB: Schedule packet H on link BC, and packets $E, F \oplus H$ on link CB. Size of H is $1 - m$, and size of $E, F \oplus H$ is $1 - l$. Thus, we are within capacity t and s of links BC and CB, respectively (as per inequalities 9 and 8, respectively).

  Recall from inequality 15 that $t + s \geq 1$. Thus there is capacity left on links BC and CB together to transmit packets D, G, I. Schedule parts of D, G, I on BC, and rest on CB such that all of D, G, I is scheduled, and the capacity of links BC and CB is not exceeded. Recall that both B and C receive D, G, I from S.

  Observe that B and C either exchange a certain packet (from D, E, F, G, H, I) or enough information to infer the packet. In particular, exchanging H and $F \oplus H$ allows inferring F. We will see later how this is useful.

- Links AC and CA: Schedule packet F on link AC, and packets $G, F \oplus H, I$ on link CA. Size of F is $1 - m$, and size of $G, F \oplus H, I$ is $1 - k$. Thus, we are within capacity u and q of links AC and CA, respectively (as per inequalities 10 and 6, respectively).

  Recall from inequality 16 that $u + q \geq 1$. Thus there is capacity left on links BC and CB together to transmit packets D, E. Schedule parts of D, E on AC, and rest on CA such that all of D, E is scheduled, and the capacity of links AC and CA is not exceeded.

Observe that A and C either exchange a certain packet (from D, E, F, G, H, I) or enough information to infer the packet.

Round 3: Assessment The peers use the information they have received from S and other peers to determine whether there is a "mismatch" (or inconsistency) between their information and the information of the peers. This process may also include further communication with S, as elaborated below. In case of a detected inconsistency, a peer may accuse another peer of being faulty. These accusations may not be accurate, since the source node S may send incorrect information to the peers, making a peer appear faulty, and a faulty peer may not necessarily send inconsistent information.

Each assessment is either f or g. At time 0, at any peer, the initial assessment of other nodes is g (good or fault-free). Once any assessment changes to f (faulty), it remains f, even when the algorithm is executed multiple times (that is, assessment are initialized to g only at time 0, and not re-initialized at the start of each execution of the algorithm). Thus, in the following, we only describe the instances where an assessment is made equal to f.

We now explain how each peer determines whether to assess another peer as faulty (f). Let us consider the assessment at each peer.

- Assessment at node A: We describe checks performed for each of the packets.

  Packet D: Node A has exchanged all the bits of D with B over links AB and BA together. Node A compares the bits of D that it receives from B on link BA with the corresponding bits received from S, and if there is a mismatch, B is assessed as faulty. Similar check for bits of D is performed for node C.

  Packet E: Node A has exchanged all the bits of E with C over AC and CA together. Node A compares the bits of E that it receives from C on link CA with the corresponding bits received from S, and if there is a mismatch, C is assessed as faulty.

  Packets F, H: Node A has received H from B, and $F \oplus H$ from C. We will use subscripts below to denote the node from which a certain value is received. For instance, $(F \oplus H)_C$ will denote $F \oplus H$ received from C, and F_S will denote F received from S. Node A computes H as *computed* $H = (F \oplus H)_C \oplus F_S$. If the computed H mismatches with H_B, then node A transmits H_B to node S on link AS. Node S responds using a single bit to indicate whether H_B is correct (if node S is faulty, it may respond arbitrarily). If node S responds that H_B is incorrect, then node A assesses that B is faulty, otherwise A assesses that C is faulty.

  Packets G, I: Node A has received packets G, I from nodes B and C both. If the two copies match, no change is made to the current assessment of B and C. In case of mismatch, G_B, I_B is sent to S on link AS. S responds with 1 bit whether this value of

G_B, I_B is correct or not. If S responds that it is incorrect, node A assesses node B as faulty, otherwise assesses C as faulty.

Observe that, on link AS, the maximum amount of data sent in round 3 is $|G, H, I| = (m - l) + (1 - m) + (l - k) = 1 - k$. Observe that link AS is used only if a mismatch of packets is detected above, and node S sends the feedback only if packets are sent to S on link AS.

- Assessment at node B: The procedure at node B is similar to that at node A.

  Packet D: This check is analogous to that at node A, and may result in node A or C assessed faulty (in case of mismatch).

  Packet E: Node B has received E from node A and node C. If the two copies match, no change is made to the current assessment of A and C. In case of mismatch, the copy of E received from A is sent to S on link BS. S responds with 1 bit whether it considers that copy correct or not. If S responds that the copy is incorrect, node B assesses node A as faulty, otherwise assesses C as faulty.

  Packets F, H: Node B has received F from A, and $F \oplus H$ from C. Using $F \oplus H$ received from C, and H received from S node B computes F. If this computed F mismatch with F_A, then node B transmits F_A to node S on link BS. Node S responds using a single bit to indicate whether that copy of F is correct (if node S is faulty, it may respond arbitrarily). If node S responds that the copy is incorrect, then node B assesses that A is faulty, otherwise B assesses that node C is faulty.

  Packets G, I: Node B has exchanged all the bits of G, I with C over links BC and CB together. Node B compares the bits of G, I that it receives from C with the bits it received from S, and if there is a mismatch, C is assessed as faulty.

  Observe that, on link BS, the maximum amount of data sent in round 3 is $|E, F| = (m - l) + (1 - m) = 1 - l$. Also observe that link BS is used only if a mismatch of packets is detected above, and node S sends the feedback only if packets are sent to S on link BS.

- Assessment at node C:

  Packet D: This check is analogous to that at node A, and may result in node A or B assessed faulty (in case of mismatch).

  Packet E: Bits of E received from S are compared with those received from A, and a mismatch results in node A being assessed as faulty.

  Packet F, H: Node C has received F from A and $F \oplus H$ from S. Using $F \oplus H$ received from S, and F received from A, node C computes H. If this computed H mismatch with the H received from B, then node C transmits H received from B to node S on link CS. Node S responds using a single bit to indicate whether that copy of H is

correct (if node S is faulty, it may respond arbitrarily). If node S responds that the copy is incorrect, then node C assesses that B is faulty, otherwise C assesses that node A is faulty.

Packets G and I: Node C has exchanged all the bits of G, I with B over links BC and CB together. Node C compares the bits of G, I that it receives from B with the bits it received from S, and if there is a mismatch, B is assessed as faulty.

Observe that, on link CS, the maximum amount of data sent is $|F| = 1 - m$. Also observe that link CS is used only if a mismatch of packets is detected above, and node S sends the feedback only if packets are sent to S on link CS.

Recall that communication between one or more peers and node S occurs during the assessment *only if* a mismatch of packets is detected at one of the peer nodes. If the network is still in mode I, then no mismatch will be detected, and no such communication will be necessary. However, when a mismatch is detected, additional communication between one or more peers and S is needed, and additional time must be allocated to round 3. This optional communication can be achieved as follows.

In round 3, each peer first determines whether it needs to perform communication with node S for the purpose of assessment or not, and sends a 1-bit message to the other peers informing them of the determination. To ensure consistent dissemination of these messages, each peer delivers its 1 bit message to other peers and S using a traditional Byzantine agreement algorithm, such as the algorithm by Pease, Shostak and Lamport [33], [22]. Since at most one node is faulty, agreement on these messages can be ensured. This allows all the fault-free nodes to synchronously add time to round 3.

Each peer is allowed to request additional time for round 3 *only once*. This ensures that a faulty node cannot needlessly keep adding time to round 3. Any peer asking for additional time more than once will be determined to be faulty. In this event, the faulty node can be excluded from the algorithm, and the rest of the nodes treated (correctly) as fault-free.

How much time should be added to round 3? We assume that the capacity of links AS, BS and CS is > 0. Since a finite number of bits need to be exchanged during the extended period, with a non-zero capacity, the duration required will be finite as well. Also, round 3 is extended only a finite number of times, and each such extended round only requires a finite amount of time. As we will see later, in a pipelined execution, the overhead of round 3 amortized over duration t approaches 0 as $t \to \infty$.

The overhead of the Byzantine agreement over the 1 bit messages results in a normalized cost of $O(1/R)$. By scaling R we can make this overhead arbitrarily small (as discussed later).

Why is it acceptable to allow each peer to extend round 3 only once? The purpose of such an extension is to allow a peer to determine which one of the other peers may be faulty.

For instance, suppose that node A sends to node S packets G, H, I and receives feedback from S that suggests that node B may be faulty. It should be easy to see that there is no disadvantage in sending these packets to S again in the future. If the feedback from S was accurate, then it will remain unchanged. If the feedback from S is incorrect, it is possible that node S will stick to this incorrect feedback in the future as well. Thus, in each case, the execution may evolve such that the feedback from S remains unchanged, and therefore, we can eliminate the communication between A and S altogether, after communicating once.

Round 4: Agreement By the end of round 3, each peer has obtained its own assessment of other peers. Each peer transmits its assessments to other peers and S using the traditional Byzantine agreement, such as the algorithm by Pease, Shostak and Lamport [33], [22]. Since at most one node is faulty, all nodes obtain identical information about the assessments by other nodes. This allows all the fault-free nodes to be in an identical mode at the start of each round.

Since the assessment sent by each node is 2 bits, the normalized cost of each node's assessment is $2/R$. The overhead of performing Byzantine agreement for these assessments is at most $6/R$ or $O(1/R)$ per link. By scaling R, this overhead can also be made small, as seen later.

When a peer node receives the assessments from other peers, it forms a diagnosis graph. An example of such a graph is shown in Figure 3. Due to the use of Byzantine agreement for the assessments, all fault-free peers will form identical diagnosis graphs. In Figure 3, we see that node A has assessed node B as faulty, but has not changed the initial g assessment of node C. The notion of diagnosis here is borrowed from past literature on *system-level diagnosis*, including the seminal paper by Preparata, Metze and Chien (PMC) [35]. The diagnosis graph used here is also similar the of *conflict graph* used in prior work on consensus [37].

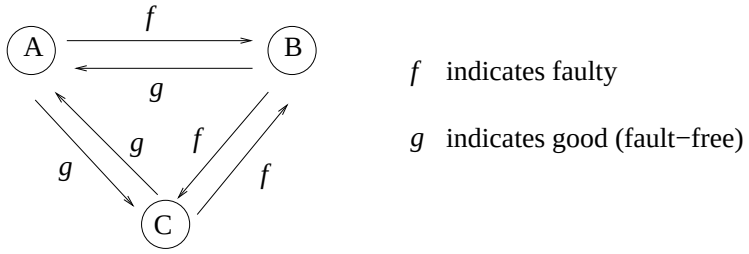

Figure 3: Diagnosis graph

Using the diagnosis graph, each fault-free peer determines which data it should agree on such that the correct agreement with other fault-free peer(s) is ensured. We will elaborate on this in the following.

- *Mode I*: If none of the edges of the diagnosis graph are f, then all nodes are operating correctly (as shown in Claim 1), and each peer node agrees on the data it can infer using the information received from S and its peers. In particular, node A has received D, E, F from S and G, H, I from B, and node A agrees on these 6 packets. Node B received D, G, H, I from node S, and E, F from node A. Similarly, node C received D, E, G, I from S, H from B, and F from A. Thus, all the nodes are able to obtain all the data.

  In this case, since no failure has been detected, the network continues operating in Mode I.

Claim 1 *There is no f in the diagnosis graph if and only if (iff) no node has behaved incorrectly.*

Proof: If all nodes are non-faulty, it is easy to see that there will be no f in the diagnosis graph.

Now suppose that there is no f in the diagnosis graph. If one of the peers nodes are behaving incorrectly:

- Node A: First of all, node A cannot misbehave by making faulty assessments about B or C, otherwise there will be at least one f in the diagnosis graph. Then the only way A can misbehave is to send some "bad" bits to B or C. Suppose A corrupts some bits it sends to B: If A corrupts bits of D, since B has received the whole correct packet D from S, it will detect a mismatch and then will assess A as faulty. If A corrupts bits of E or F, since B has received the correct packet H from S, and $E, F \oplus H$ from C which is fault-free, B can recover the correct packet E, F and compare them with the ones from A and detect a mismatch. Then B will send E and F it received from A to S and assess A as faulty upon receiving S's reply. So A cannot corrupt the bits it exchanges with B. A similar argument can show that A cannot corrupt the bits it exchanges with C.
- The argument for node B or C being faulty is similar.

If node S is behaving incorrectly. This means that node S sends inconsistent packets to the peers.

- D: since D is exchanged between every peer pair, at least two peers will see a mismatch;
- E: S transmits E to A and C and all bits of E are exchanged between A and C, at least one of them will see a mismatch;

- F, H: S transmits F to A, H to B and $F \oplus H$ to C. If $F_A \oplus H_B \neq (F \oplus H)_C$, then A, B, and C will all see a mismatch will result in 3 f's in the diagnosis graph. If $F_A \oplus H_B = (F \oplus H)_C$, then it is as if S did not corrupt any one of these three packets, since the peers will form consistent view of these packets.
- G, I: S transmits G, I to B and C and all bits of G, I is exchanged between B and C, so both of them will see a mismatch.

So S cannot transmit inconsistent data to the peers without causing some edge(s) in the diagnosis graph becoming f. □

- *Mode II:* If one or both edges between only one of the node pairs in the diagnosis graph is f, then the network is in Mode II. In this case, it can be argued that the third peer must be fault-free (as implied by Claim 2):

Claim 2 *An edge in the diagnosis graph is f only if either node S or one of the peer nodes associated with the f edge is faulty.*

Proof: Recall that at most one node in the network may be faulty. Suppose that an edge between A to B is f, and suppose in contradiction that S, A and B are all fault-free. Thus only node C may be faulty. Since S is fault-free, the packets A and B received from S are correct and consistent. The D, E, G, I packets A and B have exchanged are also correct since they are not faulty. So regardless of whether C has sent incorrect data to A and/or B or not, A and B will not assess each other as faulty, and the edges between A and B in the diagnosis graph will remain g. This is a contradiction. Similar argument can be used for packets F, H as well.

The proof is similar for other edges. □

Such a situation is illustrated in Figure 4. In this example, one of nodes S, A and B must be faulty, as per Claim 2. Therefore, due to the single fault assumption, node C must be fault-free.

It is important that node C can be argued to be fault-free, since we can now require the fault-free peers to agree with C. This is achieved as follows:

Recall that nodes A and C have exchanged packets $D, E, I, G, F, F \oplus H$ on links AC and CA together. Similarly, B and C have exchanged $D, E, G, H, I, F \oplus H$ with each other on links BC and CB. A and B have assessed C as fault-free, and vice-versa. Thus, the H computed by C and H sent by B must be identical. We require C to agree on this value of H, and the F value sent by A. Similarly, B (if fault-free) is required to agree on its own value of H, and F computed using own value of F and $F \oplus H$ received from C. Similarly, A (if fault-free) is required to agree on its own value of F and H computed using $F \oplus H$ received from C. It should be clear that, given the diagnosis

graph, the fault-free peers will agree on identical F and H. Secondly, if S is fault-free, these values will be identical to those sent by S, since the agreed values are consistent with $F \oplus H$ received by C, and the F or H value sent by A or B, respectively.

The above paragraph shows how the fault-free peers agree on correct F and H. Node C agrees on the local copies of D, E, G, I (that is, packets received from S).

Now consider node B, if it is fault-free: Node B agrees on E received from node C. Also, node B has exchanged remaining packets (D, G, I) with C; since no mismatches were detected, these packets at B must be identical to those at C, and therefore B agrees on the local copy of D, G, I.

Consider node A, if it is fault-free: Node A agrees on G, I received from C. from node C. Also, node A has exchanged remaining packets (D, E) with C; since no mismatches were detected, these packets at A must be identical to those at C, and therefore A agrees on the local copy of D, E.

The above argument shows how the fault-free peers can agree correctly provided that the only f links are between nodes A and B. As noted above, this implies that node C is fault-free.

Similar argument can be used to show correct agreement in cases (a) and (b) below also.

(a) one or both edges between only A and C in the diagnosis graph are f (implying that B is fault-free):

- Node A has sent F to node B, and node C has sent $F \oplus H$ to node B. Since B has not accused either of A and C of being faulty, these F and $F \oplus H$ values must be consistent with the H value at node B. Node B will agree on the local value of H, and F received from A. Node A (if fault-free) can correctly agree on H received from B and the local value of F. Node C (if fault-free) can correctly agree on the H received from B, and F computed using this H and the local value of $F \oplus H$.
- B has received packet E from both A and C and no mismatch was detected (in case of a mismatch, B would have accused A or C of being faulty). Node B agrees on the received copy of E, and nodes A and C (if fault-free) agree on local copy of E.
- Node B has exchanged packet D with both A and C and no mismatch was identified. Node B agrees on local copy of D. Nodes A and C (if fault-free) are able to correctly determine that their local copy of D matches with B's local copy, and therefore, agree on their local copy of D.
- Node B agrees on local copy of G, I. Node A (if fault-free) agrees on G, I received from B. Nodes B and C have exchanged G, I, and not identified a mismatch. Node C (if fault-free) agrees on its local copy of G, I.

(b) one or both edges between only B and C in the diagnosis graph are f (implying that A is fault-free): This case is similar to above case.

- Node A has received packet H from B, and $F \oplus H$ from C. Since no inconsistency with the copy of F at node A was detected by node A, node A will agree on H received from B, and F received from S. Node B (if fault-free) will agree on local copy of H, and F received from node A. Node C (if fault-free) will agree on F received from node A, and H computed using F from A and the local copy of $F \oplus H$.
- Node A has exchanged packet D with both B and C and no mismatch was identified. Node A agrees on local copy of D. Nodes B and C (if fault-free) are able to correctly determine that their local copy of D matches with A's local copy, and therefore, agree on their local copy of D.
- Node A agrees on local copy of E. Node B (if fault-free) agrees on E received from A. Nodes A and C have exchanged E, and not identified a mismatch. Node C (if fault-free) agrees on its local copy of E.
- Node A has received packets G, I from both B and C, and no mismatch was detected. Node A agrees on the received copy of G, I, and nodes B and C (if fault-free) agree on their local copy of G, I.

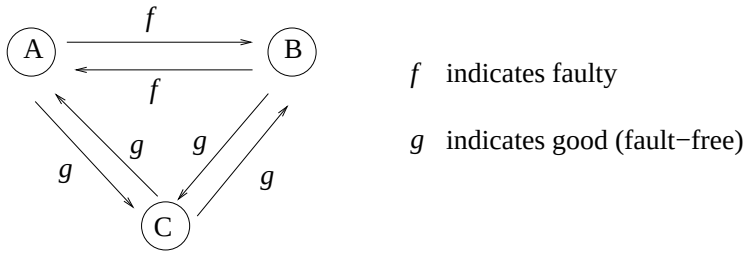

Figure 4: Mode II example

- *Mode III:* If one or both edges between two node pairs have become f, then the network is now in Mode III. We provide a discussion of mode III below.
- *Mode IV:* If one or both edges between all three node pairs are f, then the network is now in Mode IV. In this case, according the Claim 2, the faulty node must be in $\{A,B,S\} \cap \{B,C,S\} \cap \{A,C,S\} = \{S\}$. Thus, the fault-free peers can correctly infer that node S must be faulty, and agree on some default value (such as $\perp$), and terminate. No further rounds are necessary as such, since S has been diagnosed as faulty.

Observe that in modes I, II, III, and IV, the f edges are confined to 0, 1, 2, and 3, node pairs, respectively. Since f edges never change back to g, it follows that only the

following mode changes may occur: mode I may only change to mode II, III or IV; mode II may only change to mode III or IV; mode III may only change to mode IV.

4.3 Operation in Mode III

The network is in mode III when edges in the diagnosis graph between two node pairs are f, and the edges between the third node pair are both g. From Claim 2, it follows that the nodes in the third node pair must be both fault-free. Thus, the remaining peer node cannot be trusted to provide correct data (it may be faulty, or S may have provided it bad information). Consider the example in Figure 3. According to Claim 2, the faulty node must be in $\{A,B,S\} \cap \{B,C,S\} = \{B,S\}$. Thus, in this case, nodes A and C are surely fault-free.

If S is fault-free as well, then the information A and C received from S is correct. In any case, nodes A and C can agree with each other using information they have sent to each other, as follows:

- Nodes A and C have exchanged D, E with each other, and found them to match. Nodes A and C both agree on their local copy of D, E.
- Node A agrees on its local copy of F, and node C agrees on F received from A. Node A agrees on H computed using local copy of F and $F \oplus H$ received from C. Similarly, node C agrees on H computed using local copy of $F \oplus H$ and F received from A.
- Node C agrees on local copy of G, I, and node A agrees on G, I received from node C.

Clearly, the above procedure allows nodes A and C to agree on identical packets. If node S were to be fault-free, these values would also be consistent with packets sent by node S to nodes A and C.

In this scenario, node B may not necessarily be faulty (that is, the faulty node may be node S). Thus, we need to provide node B the opportunity to also agree with nodes A and C. To facilitate this, once the network enters mode III with nodes A and C being diagnosed as surely fault-free (that is, failure known to be limited to {S,B}), we change the communication schedule on links AB and CB. The goal here is to allow nodes A and C to transmit the values that they agree on to node B. The algorithm changes as below once the network enters mode III, with nodes A and C known to be fault-free.

- Round 1: Remains same as presented previously.
- Round 2: Schedule on links AC, BC, BA and CA remains same as before, but no transmissions are performed on links AB and CB in round 2.

- Round 3: Node B (if fault-free) does not perform any new assessments anymore. Other than that, the procedure remains the same as Round 3 presented previously.

- Round 4: Similar to the previous round 4, nodes A and C distribute their new assessments to all the nodes using Byzantine agreement. If the diagnosis graph remains in mode III after this, nodes A and C both compute agreed packets as discussed above. In the original round 4, after the assessments have been distributed, each peer only performs local computation. However, in mode III, now nodes A and C together transmit all the *agreed* packets D, E, F, G, H, I to node B on links AB and CB. Note that links AB and CB have not been used yet in the above steps for mode III, and from inequality 12, we have $r + s \geq 1$. Thus, there is enough capacity on links AB and CB together to transmit all agreed packets to B (nodes A and C already have all the agreed packets).

  If the diagnosis graph changes to mode IV, however, then node S must be faulty, and the peer nodes agree on some default value (say, $\perp$) and terminate.

The above modified algorithm is useful when nodes A and C are known to be fault-free, with the detected fault being confined to {B,S}. Similar changes can be made to the algorithm in the other two cases of mode III, that is, when fault is known to be confined to {A,S}, or confined to {C,S}.

4.4 Throughput Analysis

Observe that, with the exception of the determination and dissemination of the faulty assessment (in rounds 3 and 4), each link is used in only 1 round of the algorithm. Also, ignoring the overhead of computing and disseminating the assessments, each link is used for 1 time unit over all the rounds combined (this conclusion follows by adding up the size of packets sent on each link during all the rounds). The 2 bit response from node S (in round 3) and the dissemination of assessments (in round 4) requires a fixed number of bits per link, independent of R. Thus, the total time overhead for these operations is $O(1/R)$, and by increasing R (which can be achieved by scaling the unit of time), it is possible to make the overhead of diagnosis messages arbitrarily close to zero. This allows us to achieve throughput arbitrarily close to 1, as follows.

In achieving rate close to 1, 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 1 unit starts per "clock cycle" (where the clock cycle of the pipeline is long enough for each of the rounds), as shown in Figure 5. Each generation consists of four rounds and the packets are scheduled according to the schedules we discussed in the previous section, depending on which mode the system is in. At the end of the round 3 of every generation, node S sends the 2 bit indications

to the peers. At the beginning of the round 4, the peers exchange their own assessments and determine the new mode of the system. The time duration required for data transmission in each round is at most 1 time unit (recall that no link carries more information packets than its capacity permits). The assessment and dissemination operations require $O(1/R)$ time, which can be made small by choosing a large R.

Thus, we can make the duration of each round to be equal to $1 + O(1/R)$. Since a new generation of 1 unit worth of information is initiated in each round, it follows that the agreement throughput is $1 - O(1/R)$. Thus, by scaling the time unit, the throughput can be made arbitrarily close to 1.[4]

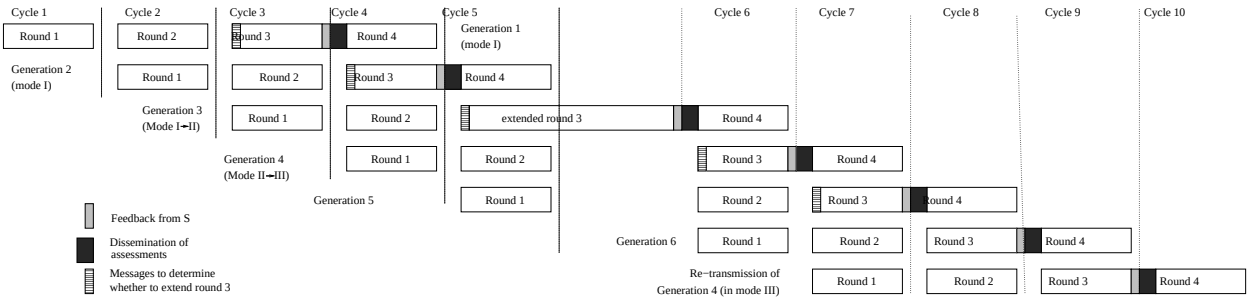

Figure 5: Example of pipelining: In generation 2, node A assesses B faulty and the system enters mode II. In generation 3, node C assesses B as faulty and the system enters mode III; generations 3, 4 and 5 are then dropped and retransmitted after entering mode III.

Figure 5 shows an example execution of the pipelining. The system starts in mode I. Nominally 1 clock cycle is assigned to each round, including round 3. When round 3 is not extended, much of the time in round 3 would be idle time. The system is in mode I for generations 1 and 2. During round 3 of generation 3, node A decides that it needs to communicate with S, and the round 3 is extended as shown. By the end of this round (cycle 5), node A assesses that node B is faulty, and the system enters mode II after the assessments are exchanged at the beginning of round 4 of generation 3. In round 3 of generation 4, node C assesses node B as faulty. Since B has already been accused by A, the system now enters mode III after the assessment exchange in round 4 of generation 4 (nodes A and C are identified as faulty, with the detected fault being confined to {B,S}). Note that, at the time the system enters mode III, three generations of packets are in the system

[4]In the special case of $k = 1$, the throughput can be made exactly equal to 1, by eliminating the need for the assessment messages. In this case, the algorithm reduces to the agreement algorithm proposed by Lamport, Shostak and Pease [22].

using the old schedule (in this example: generations 4, 5 and 6). To allow a transition to the new schedule in mode III, the packets belonging to generations 4, 5 and 6 are dropped, and retransmitted using the algorithm/schedule for mode III. Thus, agreement for the dropped generations is re-initiated in subsequent clock cycles.[5]

Observe that with the above pipelined operation, each link is used for data transmission at most once in each round, no matter how the mode changes. This implies that all the links are being used within their capacity constraint. Although three generations may be dropped when the system enters mode III, the overhead of dropped generations approaches zero asymptotically (as time t approaches ∞). Recall that the transition to mode III occurs only once. Similarly, the constant duration overhead (constant, independent of R) of extended round 3 also occurs only a finite number of times – thus, the normalized overhead of extended round 3 decreases as R increases. Thus, the normalized throughput approaches 1 asymptotically despite the dropped generations, as $t \to \infty$ and $R \to \infty$.

5 Lower Bound on Capacity

A lower bound on capacity follows from the above algorithm and its capacity analysis, and the inequalities presented previously. In particular, a lower bound on agreement capacity is obtained by taking the *minimum* over the left hand sides of all the inequalities 2 through 13.

6 Conclusion

This report defines throughput and capacity of Byzantine agreement, and illustrates the concepts through the example of a four node network.

A future report will describe our more recent results on this topic.

Many problems remain open, including generalization of the agreement algorithm to other topologies and larger number of failures, and a complete capacity characterization.

Acknowledgements

We thank Jennifer Welch for pointing us to the past work on continuous consensus and multi-Paxos. Thanks are due to Pramod Viswanath for a discussion that helped us remove an unnecessary constraint previously included in Section 4.1.

[5] In fact, it is not essential that generation 6 be dropped. We drop it here to simplify the discussion.

Research reported here is supported in part by Army Research Office grant W-911-NF-0710287. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

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

[2] N. Cai and R. W. Yeung. Network error correction, part II: Lower bounds. *Communications in Information and Systems*, 6(1):37–54, 2006.

[3] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[5] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. *ACM Symposium on Principles of Distributed Computing*, 2007.

[6] M. Chereque, D. Powell, P. Reynier, J. Richier, and J. Voiron. Active replication in Delta-4. In 22^{nd} *International Symposium on Fault-Tolerant Computing*, pages 28–37, July 1992.

[7] E. C. Cooper. Replicated distributed programs. In *ACM Symp. on Oper. Syst. Princ.*, pages 63–78, 1985.

[8] L. Davidovitch, S. Dolev, and S. Rajsbaum. Stability of multivalued continuous consensus. *SIAM J. Comput.*, 37(4):1057–1076, 2007.

[9] C. Dwork and Y. Moses. Knowledge and common knowledge in a byzantine environment: crash failures. *Inf. Comput.*, 88(2):156–186, 1990.

[10] C. Fragouli, D. Lun, M. Mdard, and P. Pakzad. On feedback for network coding. In *in Proc. of 2007 Conference on Information Sciences and Systems (CISS)*, 2007.

[11] R. Friedman, A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Distributed agreement and its relation with error-correcting codes. In *DISC '02: Proceedings of the 16th International Conference on Distributed Computing*, pages 63–87, London, UK, 2002. Springer-Verlag.

[12] C. Gkantsidis and P. Rodriguez Rodriguez. Cooperative security for network coding file distribution. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–13, April 2006.

[13] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. R. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *IEEE International Symposium on Information Theory*, 2004.

[14] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard. Resilient network coding in the presence of byzantine adversaries. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 616–624, May 2007.

[15] D. C. Kamal, D. Charles, K. Jain, and K. Lauter. Signatures for network coding. In *Fortieth Annual Conference on Information Sciences and Systems*, 2006.

[16] S. Kim, T. Ho, M. Effros, and A. Salman. Network error correction with unequal link capacities. In *47th Annual Allerton Conference on Communication, Control, and Computing*, October 2009.

[17] R. Koetter and M. Mdard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11:782–795, 2001.

[18] 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.

[19] R. Kotla, A. Clement, E. Wong, L. Alvisi, and M. Dahlin. Zyzzyva: speculative byzantine fault tolerance. *Commun. ACM*, 51(11):86–95, 2008.

[20] M. N. Krohn. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 226–240, 2004.

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

[22] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[23] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. Prog. Lang. Syst.*, 4(3):382–401, July 1982.

[24] Q. Li, D.-M. Chiu, and J. Lui. On the practical and security issues of batch content distribution via network coding. *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, pages 158–167, Nov. 2006.

[25] S. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49:371–381, 2003.

[26] G. Liang, R. Agarwal, and N. Vaidya. Secure capacity of wireless broadcast networks. *Technical Report, University of Illinois*, September 2009.

[27] G. Liang, R. Agarwal, and N. Vaidya. When watchdog meets coding. In *INFOCOM 2010 (to appear)*, 2010.

[28] G. Liang and N. Vaidya. When watchdog meets coding. *Technical Report, University of Illinois*, May 2009.

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

[30] G. M. Masson, D. M. Blough, and G. F. Sullivan. System diagnosis. In D. K. Pradhan, editor, *Fault-Tolerant Computer System Design*. Prentice Hall, 1996.

[31] T. Mizrahi and Y. Moses. Continuous consensus via common knowledge. In *TARK '05: Proceedings of the 10th conference on Theoretical aspects of rationality and knowledge*, pages 236–252, Singapore, Singapore, 2005. National University of Singapore.

[32] T. Mizrahi and Y. Moses. Continuous consensus with failures and recoveries. In *DISC '08: Proceedings of the 22nd international symposium on Distributed Computing*, pages 408–422, Berlin, Heidelberg, 2008. Springer-Verlag.

[33] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JOURNAL OF THE ACM*, 27:228–234, 1980.

[34] F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16(6):848–854, 1967.

[35] F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Trans. Electr. Comput.*, (6):848–854, December 1967.

[36] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36:335–348, 1989.

[37] C. C. with Ambiguous Failures. Tal mizrahi and yoram moses. *Distributed Computing and Networking (Lecture Notes in Computer Science)*, 4904/2008, 2008.

[38] R. W. Yeung and N. Cai. Network error correction, part i: Basic concepts and upper bounds. *Communications in Information and Systems*, 6(1):19–36, 2006.

[39] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1409–1417, April 2008.

[40] F. Zhao, T. Kalker, M. Medard, and K. J. Han. Signatures for content distribution with network coding. In *IEEE International Symposium on Information Theory (ISIT)*, 2007.