CAP for Networks

Aurojit Panda[†] Colin Scott[†] Ali Ghodsi[†*] Teemu Koponen[♮] Scott Shenker[†◇]
[†] *UC Berkeley* [*] *KTH/Royal Institute of Technology* [♮] *VMware* [◇] *ICSI*

Alice laughed. "There's no use trying," she said: "one can't believe impossible things."

"I daresay you haven't had much practice," said the Queen. "When I was your age, I always did it for half-an-hour a day. Why, sometimes I've believed as many as six impossible things before breakfast."

(Lewis Caroll)

ABSTRACT

The CAP theorem showed that it is impossible for datastore systems to achieve all three of strong consistency, availability and partition tolerance. In this paper we investigate how these trade-offs apply to software-defined networks. Specifically, we investigate network policies such as tenant isolation and middlebox traversal, and prove that it is impossible for implementations to enforce them without sacrificing availability. We conclude by distilling practical design lessons from our observations.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Network operating systems

Keywords

Software Defined Network, Distributed Controllers, Correctness, Availability

1. INTRODUCTION

In his famous PODC keynote [5], Eric Brewer articulated the CAP conjecture, a fundamental trade-off between linearizability,[1] availability and partition tolerance in distributed database systems. The CAP conjecture and its subsequent proof [10] strongly influenced the design of distributed storage systems, especially recent 'NoSQL' designs. In this paper we argue that a similar set of trade-offs and choices apply to the control algorithms used in networks.

While availability and the ability to withstand network partitions has long been an important goal in networking [7], prior literature has not analyzed the impact of this choice on enforceable policies. Networks support increasingly complex policies, making it vital that network architects be aware of the trade-offs between policy enforcement and partition tolerance. To the best of our knowledge, this is the first study on the enforceability of network policies in the face of network failure.

Our investigation is motivated by the recent move towards software-defined networking. Software-defined networks move control plane functionality out of switches and into separate controllers. Controllers in these networks typically communicate through an out-of-band management network to coordinate among themselves [1].

The crucial consequence of out-of-band control is that the network may enter a situation where the controllers are partitioned from each other while the data network remains connected. In such a scenario, network policies that would otherwise be implementable may be violated.[2] Further, such partitions, which can be a result of both physical failures and software bugs, are prevalent in practice [14]. Lastly, network services that are available in the presence of partitions cannot rely on global coordination, and therefore availability in practice implies better performance [2]. In this paper we seek to identify the set of policies controllers can and cannot be achieved under partition.

While availability and partition tolerance are identical in networks and data stores, the notion of consistency differs significantly. In storage systems the read and write semantics of data across replicas is the primary concern. In SDN networks, by contrast, we are interested in consistent application of policies across the network. The definition of consistency in networks therefore depends on the precise policy under consideration. In this paper we consider isolation policies, middle box processing, and traffic engineering, and present impossibility results for these policies.

[1] Brewer's original talk referred to consistency, without specifying a particular form of consistency. However in subsequent work, the wider distributed systems community defines the consistency level specified by CAP to be linearizability, or atomic consistency [10]. Recent work [18] has shown that weaker forms of consistency (e.g. causal consistency) do not require trading off on availability or partition tolerance.

[2] We discuss in-band and out-of-band control in greater detail in §6.

HotSDN'13, August 16, 2013, Hong Kong, China.

ACM 978-1-4503-2178-5/13/08 ...$15.00.

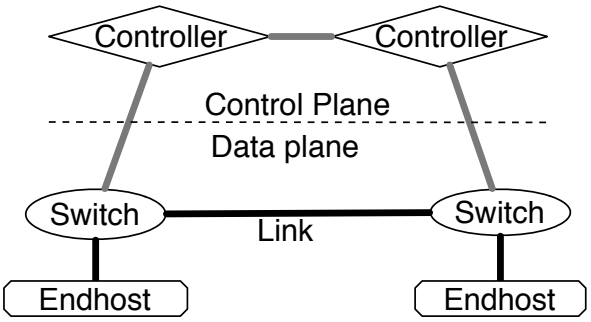

Figure 1: Entities in the Network Model

Similar to other impossibility proofs [9, 10], we do not intend to imply that practitioners should not implement these policies; rather, the main value of our proofs is to elucidate points in the trade-off space so practitioners can make informed choices.

2. MODEL

In this paper we wish to prove assertions that hold over all possible controller algorithms and all possible network topologies. Rather than reasoning about the full detail of networks, we present a simple model that retains the core properties we are attempting to reason about.

The entities in our model include a set of SDN controllers, switches that implement rules set by the controllers, and end-hosts. Figure 1 shows these various components graphically. We assume a separate control and data network. The data network connects switches to each other and end-hosts to switches. The control network connects controllers to each other and individual switches to controllers. End-hosts act as sources and sinks for packets in the network.

In our model, the control program is notified when a new switch connects to the controller, or a new host attaches to a connected switch. We define the *domain* of a controller to be the set of all switches directly connected to the controller and all end-hosts connected to these switches.

Interactions between switches and controllers are governed by the following set of rules:

1: Each host is associated with a unique identifier E and a topologically assigned address A. The entity identifier is persistently tied to a particular end-host, while the address associated with an end-host may change due to host migration. Packet headers and forwarding rules are specified in terms of addresses, while policies are specified in terms of entities.[3]

2: Controllers query switches and end-hosts within their domain to determine links, end-host identity and addresses.

3: Each entity belongs to exactly one controller's domain[4]. A controller cannot query an entity not in its domain and must instead contact another controller to do the querying on its behalf.

4: Controllers specify switch behavior as a set of rules, each of which references one or more addresses and an action. We allow two actions: one that forwards packets along a link and one that drops packets. While in practice switches allow for a richer set of actions, these actions are sufficient for showing the impossibility results.

In addition to these rules, we make the following assumptions throughout the paper:

- **Out-of-Band Control** The results presented here assume an out-of-band control network, where the control network is separate from the data network and the controllers never tunnel control information through the data network. While SDN networks could use both in-band and out-of-band control, many existing SDN networks [1] only use out-of-band control. We discuss the effect of switching to in-band control in §6.
- **Fail-Stop Links** We assume a fail-stop model for all links, *i.e.* we do not model link recovery, nor do we model link degradation due to partial failures. In real networks links eventually recover, but this process may take substantial time. A recent study of datacenter link failures [11] shows that failed links generally take several minutes to recover, with over 20% of failed links taking over 5 minutes to be repaired. Link repair times are thus several orders of magnitude larger than control plane convergence times and our results focus on the behavior of the network during the time when such failures persist. Moreover, our results also hold for stronger link failure models.
- **Static Policies** The controllers in our model rely on a set of operator provided policies. We assume that all controllers in a network have access to the same policy specification. For our analysis, we assume operator policies do not change. Mechanisms for consistently pushing new operator policies to the data plane have been described previously [21] and are not a focus of our analysis.
- **Policies Specified in Terms of Entities** We assume that policies are specified in terms of entity identifiers, not addresses. This practice simplifies the task of network management [6] and is particularly well suited to networks that separate location from identifiers [8]. In §6 we discuss the implications of implementing policies by instead placing constraints on addressing.
- **Host Migration** We assume that each end-host is connected to exactly one switch at a given time, yet we do allow for host migration. Host migrations are common in enterprise networks, where mobile devices might move around as a matter of course and in datacenters where VM migration is common. We assume that other than the switch notifications described previously, host migration does not necessarily involve any coordination with the network controller.
- **Proactive Forwarding** We assume that switches do not reactively consult their parent controller to make data plane decisions. We make this assumption merely for convenience to limit the number of events we have to consider in our results. Our results do not depend on this assumption and hold even in the presence of reactive control, as will become clear in the proofs.

[3]Our model is agnostic to the mechanism hosts use to resolve addresses. In practice a lookup service is often implemented by the controllers themselves [12, 20]

[4]Even if a switch is physically connected to multiple controllers, only one of the controller's is active to avoid race conditions where multiple controllers push conflicting updates to the same switch.

- **Dynamic Addresses** We assume that end-host locations and addresses can change over time without the intervention of controllers. We also assume that the address space is not statically partitioned; that is, addresses may be assigned arbitrarily.[5]

Within the model, a control application therefore operates on three kinds of information: policies provided by operators, state determined by querying directly connected switches and state obtained from other controllers. With this information the control application produces a set of rules and pushes them down to switches.

We use this model to analyze trade-offs between correctness, availability and partition tolerance in SDN networks. We provide precise definitions for these properties below:

- **Availability** In our model availability is a liveness property: packets destined to an end-host should eventually arrive as long as a path exists between the sender and the receiver and communication is not prohibited by policy. Since we assume fail-stop links (and hence that partitions can last indefinitely), availability requires that packets be delivered in the presence of partial partitions. Naturally, we do not require that packets be delivered when no physical path exists.
- **Partition Tolerance** We say a policy is partition tolerant if the network continues to operate in the presence of arbitrary partitions in the physical network.
- **Correctness** The definition of correctness used depends on the network policy. The original CAP theorem [10] defined consistency in terms of linearizable reads and writes to data objects. The consistency guarantees provided by data store systems are intended to support a wide range of applications. In the more limited context of SDN networks, there are fewer policies, and those network policies often have simpler correctness conditions. In particular, linearizability is likely unnecessary for ensuring correct application of most network policies. We therefore choose to analyze a narrower set of correctness properties, each defined by individual policies.

  Later in §5 we generalize our results by framing network consistency guarantees in terms of registers and presenting some preliminary thoughts on consistency guarantees that are sufficient to implement these policies.

3. IMPOSSIBILITY RESULTS FOR POLICIES DEPENDENT ON IDENTITY

In this section we prove that policies referencing the identity of two or more entities are not generally implementable during partitions in the control network. The basic intuition is that packets are routed on addresses, yet the mapping between addresses and entities may change even when partitioned controllers cannot receive updates. While the main result in this section is fairly obvious given our network model, it serves as an example of how one could reason about impossibility results in SDN networks, and also helps seed much of the discussion in §6.

[5]We observe that one can use static partitioning to circumvent some of our results and we make a note of this when applicable.

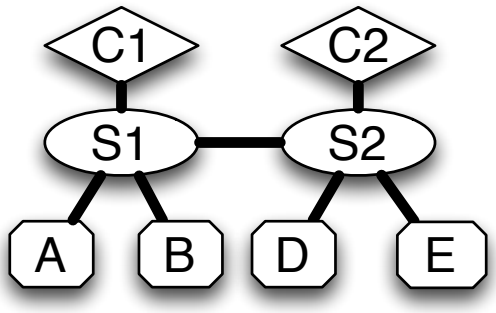

Figure 2: An example network for a violation of isolation

We start by stating some general results applicable to all such policies. Later we apply these results to two specific policies: inter-tenant isolation and middlebox traversal.

3.1 Results

A policy references an entity either when the policy statement explicitly refers to that entity, (*e.g.* 'A' and 'B' in a policy stating that end-host A may not communicate with end-host B), or when the policy depends indirectly on classes of entities (*e.g.* a policy stating that all traffic from a particular host H must be compressed depends on compression middleboxes). We ignore policies that are either vacuously unachievable (such as the previously mentioned compression policy in a network where no middlebox can perform compression), or are vacuously enforceable (such as the previously mentioned isolation policy where only one of entities A and B is connected to the network).

In the model presented in §2 routing entries are specified in terms of addresses. Controllers must therefore enforce policies by correctly resolving addresses for all entities referenced. However, during a control network partition, controllers may not correctly resolve addresses for entities outside their domain and hence cannot enforce policies referencing such entities. Below we expand this intuition into a formal proof.

Lemma 1 A controller cannot in general resolve the address for an entity outside its domain in the presence of network partitions, under the model described in §2.

Proof We prove this by showing a counterexample. Assume there exists a mechanism that allows a controller to resolve the address for an entity outside its domain, even in the presence of network partitions. Consider a network with two controllers C_1 and C_2 and entity E which belongs to controller C_2's domain. Consider applying the assumed mechanism to resolve E's addresses at controller C_1 in the presence of a control network partition between C_1 and C_2. By Rule 4, addresses are topologically assigned and can change due to a series of data plane events. Consider a situation where at some point after the control network is partitioned and entity E's address changes from A_E^0 to A_E^1. Due to Rule 3, from controller C_1's perspective this is identical to a situation where E's address instead changes from A_E^0 to A_E^2 (or remains unchanged). Since controller C_1 cannot distinguish between these cases, it cannot correctly resolve entity E's address in all cases. This is a contradiction and thus no such mechanism can exist. □

Theorem 1 All three of correctness, availability and partition tolerance cannot be achieved for policies referencing two or more entities, under the model described in §2.

Proof This is a consequence of Lemma 1. Consider a case where a policy references two entities A and B in different

domains. By Rule 4 a mechanism implementing this policy must produce one or more rules which must either (a) specify addresses for both A and B or (b) specify address for A (respectively B) in B's domain (respectively A's domain). In both these cases, a controller must resolve the address for one or more entities outside its own domain. By Lemma 1 this is not possible under controller partition and hence all three of correctness, availability and partition tolerance cannot be achieved. □

We now show that two of the three can always be achieved.

Correctness and Availability. This is equivalent to enforcing these policies in the absence of failures and is hence trivially satisfied.

Correctness and Partition Tolerance. This is trivially satisfied by sending no packets from one controllers domain to another.

Availability and Partition Tolerance. This is trivially satisfied by allowing all packets from any source to any destination. □

3.2 Impossible Policies

Next we apply the previous result to two concrete policies:

- **Isolation**: Isolation policies are used to specify that packets from a entity A can never reach another entity B. Such a policy might be useful in the context of a shared datacenter, or other cases where adversaries potentially share infrastructure.

- **Middlebox Traversal**: Middleboxes are data plane elements, commonly used to provide additional packet processing services such as compression, virus scanning, intrusion detection, or encryption. A Middlebox Traversal policy indicates that traffic from a certain source must always pass through a specific middlebox. In our model, a middlebox is functionally equivalent to a switch (*i.e.* we do not model the functional aspects of a middlebox).

3.2.1 Isolation Policy

Isolation policies, as defined previously, reference at least two entities (A and B in the example above). Therefore by Theorem 1 an arbitrary isolation policy cannot be implemented on arbitrary topologies when the control network is partitioned.

As an example of one such condition, consider the network in Figure 2 and an isolation policy requiring that A is isolated from D. Enforcing this policy requires either controller C_1 to resolve the address for entity D, or for controller C_2 to resolve the address for entity A. By Lemma 1 this is impossible under control network partitions.

3.2.2 Middlebox Traversal

Middleboxes, as stated previously, are data plane elements commonly used to provide packet processing services for packets and network transfers in a network. Middleboxes have limited processing capacity, making it essential to limit the traffic passing through them.

Middlebox traversal policies specify that packets originating from or destined to a certain entity must pass through a middlebox (and no other packets should pass through the middlebox). For instance in the network shown in Figure 3, a middlebox traversal policy might require that all traffic originating at end-host A must pass through middlebox M.

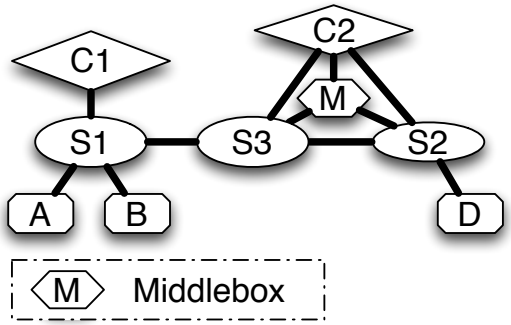

Figure 3: An example network with a middlebox.

Controllers implementing this policy must be able to resolve addresses for both entity A and middlebox M, and as shown by Theorem 1, this is impossible during control network partitions.

3.3 Workarounds

Theorem 1 is not meant to show that it is strictly impossible to implement policies referencing multiple entities in real networks. Rather, our intention is to show the precise constraints that need to be circumvented by system designers in order to work around the impossibility.

In general, when a policy references more than one entity, it is essential that a single controller be able to identify both entities unambiguously. This can either be achieved by placing both entities in the same domain, or by tagging packets with the identity of relevant entities. For instance, if a packet was labeled unambiguously with the identity of the source end-host, we would be able to achieve all of isolation, availability and partition tolerance. The practical lesson from this proof is that labels are a powerful mechanism for communicating control plane information in-band. We discuss this approach later in §6.

One can also enforce such identity based policies by constraining address allocation. As an example, consider a policy that applies to machines belonging to different tenants in a multi-tenant datacenter. One could assign each tenant a different address block (*e.g.* a unique /16 per tenant) and enforce policies over these wild-card addresses (for instance, 10.0.0.0/16 should not be able to communicate with 11.0.0.0/16). When host migration occurs, new addresses would need to assigned according to the policy. The limitation of this approach is that it restricts choices for where an end-host resides (in cases where topological addressing is used).

4. EDGE DISJOINT ISOLATION

We now consider a policy that requires that traffic between pairs of end-hosts travel along edge disjoint paths, *i.e.* that no link carry traffic sent between different pairs of end-hosts.[6] We call this property edge disjoint isolation. Such a policy might be applied in cases where competitors share network infrastructure, and observable traffic patterns can reveal information. This policy can also be considered a simple version of traffic engineering, where traffic must be load balanced across several network links.

Note that edge disjoint isolation might not always be achievable, even without partitions. We only consider situations where the policy is in fact achievable, *i.e.* can be implemented in the network's current topology.

[6]We allow packets to traverse the same switch(es).

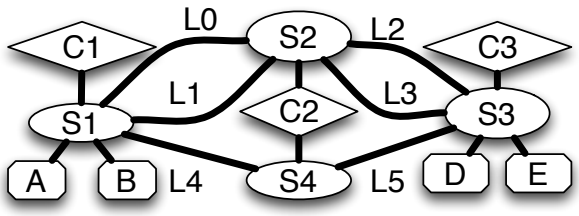

Figure 4: An example network for a violation of edge disjoint isolation

4.1 Results

Theorem 2 Only two of edge disjoint isolation, availability and partition tolerance are achievable under the model described in §2.

Proof Here we show that there exist topologies where no distributed algorithm can guarantee all three of edge disjoint isolation, availability and partition tolerance. Consider the topology shown in Figure 4 and a policy requiring that traffic from A to D be edge disjoint isolated from traffic going from B to E. Further, let us assume that there is a distributed algorithm that enforces edge disjoint isolation policies despite control network partitions.

Let us now consider a failure in link $L5$. The hypothetical algorithm must now route traffic from both A to D and from B to E through switch $S2$. However it follows from Rule 3 that from the perspective of controller $C1$, this failure scenario is identical to one where link $L3$ fails, in which case not both of A's and B's traffic can traverse $S2$. Since $C1$ cannot distinguish the two cases, its possible that both A and B's traffic would share a link, thus violating edge disjoint isolation.

It can be shown that any two of the properties can be achieved pairwise by following a similar line of reasoning as in §3.1.

4.2 Workarounds

The impossibility of edge disjoint isolation results from controllers lacking a consistent view of link failures in the network. It is easy to see that this impossibility is resolved if all controllers have consistent topology information and an agreed-upon mechanism for selecting what path a particular end-host pair should communicate over.

In our current model, topology information is exchanged over the control network. However, a variety of traditional link-state routing protocols, including OSPF [19], exchange control messages over the data plane and these mechanisms could be used to provide a consistent view of the topology. The use of traditional routing protocols is effectively a form of in-band control, which we discuss more fully in the next section.

One could also encode link failure information either in data packets, as is done in FCP [15] or in specific topologies using mechanisms based on the incoming port for a packet. Both of these methods are equivalent to tunneling control information through the data plane.

5. THE NETWORK AS REGISTERS

Consistency models in the distributed systems literature are commonly expressed in terms of the read and write behavior of registers [16]. Such models express constraints on values that can be read from a register at any point in an execution. For instance atomic (linearizable) register [13] guarantee that (i) reads return either the last value written, or the value being written by a concurrent write, and that (ii) following a read all subsequent reads return a value that is at least as recent as the value returned.

In this paper we modeled the network as an asynchronous message passing distributed system. One could instead model the routing tables in the network as a collection of registers the controllers have access to. In such a model the registers store both the set of policies that must be enforced in a network and the information required to enforce these policies. For example identity policies presented in §3 can be implemented using registers containing the current entity to address mappings for the network. Similarly, the edge disjoint policy described in §4 can be implemented using registers that can be queried to discover the network topology.

In the register model the trade-off between correctness and availability can be analyzed by determining the weakest register required to implement a given network policy. We observe that atomicity (linearizability) is sufficient to implement the policies discussed so far. Gilbert and Lynch [10] have previously shown that linearizability and availability are not achievable in the presence of partitions and consequently all three of correctness, availability and partition tolerance cannot be achieved for these policies. It is however doubtful that atomicity is the weakest consistency model necessary for implementing common network policies. We believe that determining a consistency model that is both necessary and sufficient for network policies is an important open problem.

6. DISCUSSION

In this section we attempt to distill practical design lessons from our observations.

6.1 In-Band Control

It is commonly believed that out-of-band control networks are simpler and more resilient than in-band control networks. Our findings suggest that naïve out-of-band control may actually provide lower resilience than in-band control. In particular, the impossibilities we have discussed so far essentially boil down to the inability of controllers to update their view of the network topology. With in-band control, the only time the controllers cannot update their view is when the data network is itself partitioned and the data plane operations themselves cannot be carried out.

A hybrid approach can also circumvent these impossibilities, where the controllers revert to in-band control when the out-of-band network is partitioned. Hybrid approaches provide comparable simplicity to pure out-of-band networks, while simultaneously providing greater resilience.

6.2 Labels

In-band control essentially involves tunneling control packets across physical links. Another option is to attach control information to data packets themselves. For example, the impossibility of cross-domain policies can be circumvented if the edge router labels each packet with the identity of the source. More generally, the result of any packet classification, *e.g.* the results of deep packet inspection, could be attached to packets to implement complex policies despite control plane partitions.

The extreme case of labeling is where the entire policy itself is attached to packets. This is essentially the mechanism proposed by Active Networking [22].

6.3 Consistent Network Updates

Prior work by Reitblatt et al. [21] provides a set of primitives for consistently applying planned configuration changes in software-defined networks. In particular, their work presents mechanisms for providing per-packet consistency, guaranteeing that any individual packet is processed by exactly one consistent global configuration, and per-flow consistency, guaranteeing that all packets in a flow are processed by exactly one consistent global configuration. As noted by the authors these consistency models are stronger than atomic consistency and hence the mechanisms proposed cannot be used during partitions or in the presence of other failures.

7. CONCLUSION

The main formal result in this paper is that for software-defined networks facing network partitions, several common network policies are not enforceable without sacrificing availability. Thus, SDN architects have a choice between policy enforcement and network connectivity. Traditional networks typically favor availability over policy enforcement, but the flexibility of software-defined networks allows operators to make a case-by-case decision about this trade-off depending on the specific context and desired network policies.

We have described some ways to avoid these impossibility results; that is, one can weaken the consistency model required to implement certain policies, and thereby implement these policies even in the face of partitions. This line of thought is similar to work on alternate consistency models [3, 4, 17] that allow data stores to provide weakened consistency guarantees without requiring them to trade-off availability. Our work suggests that a fuller exploration of consistency models for network policies is a fruitful area of research.

8. ACKNOWLEDGMENTS

We thank Shivaram Venkataraman, Kay Ousterhout, Peter Bailis, and Amin Tootoonchian for their feedback and suggestions. This research is supported by NSF CNS 1040838 and NSF CNS 1015459.

9. REFERENCES

[1] Big Network Controller Datasheet. Retrieved 03/22/2013: http://www.bigswitch.com/sites/default/files/sdn_resources/bnc_datasheet.pdf.

[2] D. Abadi. Problems with CAP, and Yahoo's little known NoSQL system. http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html, 2010.

[3] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. HAT, not CAP: Towards Highly Available Transactions. HotOS'13.

[4] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Bolt-on Causal Consistency. SIGMOD'13.

[5] E. Brewer. Towards Robust Distributed Systems. PODC '00 Invited Talk.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. SIGCOMM '07.

[7] D. Clark. The Design Philosophy of the DARPA Internet Protocols. CCR '88.

[8] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830.

[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. JACM '85.

[10] S. Gilbert and N. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. ACM SIGACT News '02.

[11] P. Gill, N. Jain, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. SIGCOMM '11.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. SIGCOMM '09.

[13] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. TOPLAS '90.

[14] K. Kingsbury and P. Bailis. The network is reliable. http://aphyr.com/posts/288-the-network-is-reliable, 2013.

[15] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving Convergence-Free Routing Using Failure-Carrying Packets. CCR '07.

[16] L. Lamport. On interprocess communication. Distributed Computing '86.

[17] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't Settle for Eventual: Scalable Causal Consistency For Wide-Area Storage With COPS. In *SOSP 2011*.

[18] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, Availability, and Convergence. University of Texas at Austin Tech Report '11.

[19] J. Moy. OSPF Version 2. RFC 2328.

[20] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: a Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. SIGCOMM '09.

[21] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for Network Update. SIGCOMM '12.

[22] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. CCR '96.