Practical accountability for distributed systems

Andreas Haeberlen Petr Kouznetsov Peter Druschel

Max Planck Institute for Software Systems

1 Introduction

In a distributed system with multiple administrative domains, there is generally no single entity that can monitor and control the entire system. Instead, the domain administrators must cooperatively monitor the system's health and performance, diagnose problems and effect repair. Each administrator can observe and control only the nodes in her local domain. Worse, the domains may have different and potentially conflicting interests (for example, they may be business competitors). In this case, the administrators may be reluctant to cooperate and share information.

In such an environment, a domain may not be able to assert that other domains follow the protocol and meet their contractual obligations. If something goes wrong, it may be difficult to identify the root cause and the responsible party. The domains may withhold information out of negligence, to protect business secrets, or to cover up mistakes. Even if a problem is diagnosed correctly, the responsible domain must be convinced to fix the problem. On the other hand, if the diagnosis is incorrect, a domain may be falsely implicated but unable to defend itself.

These types of problems arise, for instance, in the Internet's inter-domain routing system [8]. Here, the domains are competing network providers that exchange traffic based on bilateral peering agreements. Faults in inter-domain routing occur frequently [5]. Today, such faults are diagnosed in an ad hoc fashion, via phone calls between the various network administrators. This can lead to prolonged outages [2].

Accountability has been suggested as an important property of dependable distributed systems [9]. An accountable system produces a secure audit trail that can be inspected when a problem occurs. Accountability can be implemented from the following three building blocks:

- **Non-repudiability:** Each action is irrefutably linked to the entity that performs it. For example, nodes in a distributed system are required to cryptographically sign each message they send.
- **Secure record:** All actions are securely recorded. For example, all nodes record the actions they perform in a tamper-evident log [6].
- **Auditing:** The secure record can be inspected by other domains or by a trusted third party.

The record can be used to diagnose problems by reconstructing causal chains of events; it enables fault detection by checking a node's actions for compliance with a protocol specification; and, it allows a domain to prove that it is not responsible for an incident.

In addition, accountability discourages certain types of misbehavior. For instance, the threat of exclusion may discourage censorship and freeloading in peer-to-peer systems, and the threat of embarrassment may cause commercial service providers to improve the quality and reliability of their services.

Lastly, it is possible to balance accountability and privacy. A domain may disclose only as much information as is needed to prove its compliance, and only to its partner domains. Alternatively, domains may disclose their record to a trusted third party that audits all domains.

Whether accountability in distributed systems is practical, particularly at large scale, remains the subject of research. In this paper, we propose a type of accountability that can be achieved in practical distributed systems.

2 Practical accountability

We model a distributed system as a collection of *nodes* that run a distributed algorithm. A node can misbehave in two ways: by performing an action that is not allowed by the node's algorithm, or by failing to perform an action that the node is expected to perform according to its algorithm.

2.1 Observable misbehavior

Ideally, an accountable system detects every instance of misbehavior. Such fine-grained accounting could expose, for instance, that someone has tampered with a node's software long before the node misbehaves in an obvious way. Unfortunately, this strong form of accountability may require either trusted hardware and software attestation, or trusted probes that monitor each node's inputs, outputs, and internal memory. Neither of these requirements seem practical for large-scale systems with multiple administrative domains. Thus, it is impractical for an accountable system to generate a proof of misbehavior for *every* incorrect action.

Instead, we limit ourselves to *observable* misbehavior, that is, misbehavior that affects another correct node, directly or indirectly. In a message-passing system, there are two ways in which a correct node can be affected: by receiving a message that is causally preceded by an incorrect action of some node, or by not receiving an expected message. According to our definition of accountability, a node *appears* correct to every correct node, as long as (i) it

does not perform incorrect actions that can be observed, directly or indirectly, by a correct node, and (ii) it eventually responds to each message received from a correct node.

When a message m is received by a correct node, the message may be preceded by incorrect actions of multiple nodes. We call these nodes *faulty accomplices* with respect to m. Because a practical system cannot reliably observe messages exchanged between faulty nodes, it is impossible to identify all nodes that performed incorrect actions in this case. However, we can ensure that an accountable system generate a proof of misbehavior against one of the faulty accomplices.

2.2 Proofs and verifiable evidence

To construct an irrefutable proof of misbehavior, it is necessary to authenticate all actions, i.e., to link an action to the node that performed the action. This can be accomplished, for instance, with public-key cryptography.

However, even if all observable actions are authenticated, it is not always possible to construct a proof. The reason is that there can be *he-said-she-said* situations, e.g., when a node i claims to have sent a message m to another node j, who denies having received it. The possible reasons for the disagreement could be that (i) i has never sent m, or (ii) j has received m but denies it, or (iii) m was lost in the network. Unfortunately, a third node k cannot reliably distinguish these three cases.

We address this issue by requiring that an accountable system be able to produce *verifiable evidence* of misbehavior. Verifiable evidence against a node i consist either of (a) an irrefutable proof of i's misbehavior, or (b) a *challenge* that i did not answer. While the former can be checked independently by a third party, the latter must be checked by sending the challenge to i and by waiting for a response.

In the above example, i can use the message m as evidence against j. Any correct node k can check the evidence by challenging j with m. If j is correct, it accepts m and acknowledges it; k can then forward the acknowledgment back to i. On the other hand, if j refuses to accept m, it will not respond to k either; thus, k can convince itself that j is unresponsive. We note that the reason for the lack of a response from j could be a network problem. We discuss this issue next.

2.3 Suspicion and certainty

What if a misbehaving node creates a false challenge? For example, a faulty node i might claim that some correct node j does not respond to a message m. In a synchronous system, this is not a problem, since any other node can simply challenge j with m. If j responds, the challenge was incorrect, otherwise j is misbehaving. However, assuming synchrony is not realistic in most practical systems.

In the absence of synchrony, an unanswered challenge does not constitute a proof of misbehavior, since the response may simply be delayed. Hence, we can only *suspect* misbehavior in this case, and it is possible that the suspicion eventually turns out to be groundless.

If we make the common assumption that correct nodes can eventually communicate [1], we can guarantee that a correct node can eventually defend itself against a false challenge; thus, no correct node can be suspected forever. If we additionally assume that communication and processing are bounded [3], we can ensure that, eventually, no correct node is suspected by a correct node.

2.4 Eventual detection

How much time is needed to generate evidence against a node that misbehaves in an observable way? Ideally, the evidence would be produced as soon as the first incorrect message is received by a correct node. But a misbehaving node could send two messages that both appear plausible, but contradict each other. One way to detect this kind of misbehavior is to forward all observable messages from the misbehaving node to *some* correct node. Since instantaneous message forwarding is unrealistic, we must settle for a guarantee of *eventual* detection.

2.5 Putting it all together

Based on the above observations, we propose the following definition. A system provides accountability if, whenever a node i misbehaves in an observable way, the system eventually generates verifiable evidence against i. This evidence is either a proof of misbehavior against i (or a faulty accomplice of i) or a challenge that i did not answer.

We built a practical implementation of an accountable system with this property, called PeerReview [4]. Briefly, in PeerReview, nodes cryptographically sign all messages and commit their actions to a tamper-evident log [6]. A set of *witness* nodes inspects a node i's actions, including the set of messages that i has transmitted to correct nodes. A witness detects misbehavior by replaying i's actions using a trusted reference implementation of i's protocol. The required number of witnesses per node depends on the system's fault assumptions: it must be chosen to ensure that at least one of the witnesses is not itself faulty. PeerReview's bandwidth and CPU overhead scales linearly with the number of witnesses, while the message complexity scales with the square of the number of witnesses.

To date, we have applied PeerReview to three different systems: a distributed file system, an end-system multicast system, and a serverless email system called ePOST [7]. Our experience suggests that the system can scale to several thousands of nodes at reasonable cost, while guaranteeing eventual detection of every fault. We believe that the system can scale to even larger systems when configured to provide probabilistic detection guarantees. In current work, we are applying PeerReview to the problem of providing accountability for the Internet's inter-domain routing system based on BGP [8].

3 Conclusion

Accountability is a promising building block in the design of dependable distributed systems. In particular, it suits well systems that span multiple administrative domains. Accountability enables automatic fault localization, identifies the party responsible for a fault, discourages misbehavior, and allows correct principals to prove their compliance to third parties. Moreover, accountability can provide these guarantees while allowing domains to run different software and hardware, implement their own policies, and disclose only as much information as is needed to prove their compliance with existing agreements, protocols and standards.

Our experience with PeerReview shows that the type of accountability described in this paper is practical. In current work, we are applying PeerReview to more types of systems, and we are investigating ways to improve the system's scalability beyond thousands of nodes.

References

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

[2] J. Chandrashekar, Z.-L. Zhang, and H. Peterson. Fixing BGP, one AS at a time. In *Proc. SIGCOMM workshop on Network troubleshooting (NetT'04)*, 2004.

[3] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288 – 323, April 1988.

[4] A. Haeberlen, P. Kouznetsov, and P. Druschel. The case for Byzantine fault detection. In *Proceedings of HotDep'06*, Nov 2006.

[5] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *Proceedings of ACM SIGCOMM*, Sep 2002.

[6] P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, Jan 2002.

[7] A. Mislove, A. Post, A. Haeberlen, and P. Druschel. Experiences in building and operating ePOST, a reliable peer-to-peer application. In *Proceedings of EuroSys 2006*, April 2006.

[8] Y. Rekhter and T. Li. RFC 1771: A border gateway protocol 4 (BGP-4). Mar 1995.

[9] A. R. Yumerefendi and J. S. Chase. The role of accountability in dependable distributed systems. In *Proceedings of HotDep'05*, Jun 2005.