Causal Telemetry Project Writeup

Benny Rubin

Spring 2023

1 Problem Statement

It is notoriously difficult to get a consistent view of network state [14]. It is even harder to get a history of how state has changed over time. Networks are constantly in a state of flux with route updates, link outages, and configuration changes [1]. One of the challenges is that local clocks cannot reliably be used to order events in the network [9]. An application that could benefit from a better view into network state is the Border Gateway Protocol, responsible for routing traffic across the internet. Considering how critical BGP is to the everyday function of the internet, it is surprisingly and frighteningly delicate. BGP is highly sensitive to minute changes in configuration and can easily be misconfigured, leading to widespread network outages.

2 Motivation

Despite the fact that BGP is critical for the functioning of the modern internet, it is hardly robust in the face of updates and link outages — both Facebook and Youtube have seen serious outages from small misconfigurations [1]. Blackholes and routing loops often take on the order of minutes to fix themselves, leaving customers disconnected [11]. A single misconfigured router can propogate thousands of bad routes to its neighbors. It is clear that modern networks can be treated as large, complex, distributed systems [4].

Causal telemetry exists at the intersection of distributed systems and networking, leveraging Lamport's happened before relation and recent work on inband network telemetry to build a consistent view of network state. The high level idea is to log every event in the network, classified into three groups. Send events, receive events, and internal events. Similar to [9], sending and receiving packets can be used to order events *between* switches without the use of synchronized clocks. Using logical clocks to build causal relationships between events is a problem that has been heavily worked on in the distributed systems community [9, 2]. I believe there are a lot of fruitful insights to be discovered by applying these ideas to a networking setting.

Recent advances in programmable switches allow for behavior that was not possible due to fixed function switch architecture. These switches can add arbitrary telemetry to packets and keep logs of events in switch memory, along with a logical timestamp, to be sent to a centralized controller [6]. Leveraging this, it is possible to realistically implement causal telemetry into BGP routing. While implementing causal telemetry and logging would be difficult in the internet context, BGP is widely used within data centers, making it a rich and realistic use case for this study.

This more detailed view of network state that includes causal relationships between events can be used in a variety of ways. We have already seen it used for PFC deadlock detection and querying [10]. Sonata [5] is a powerful query language for network log data. Causal telemetry can be used identically, but with more information on the ordering of events. This can help with root cause analyses and detection of issues that depend on the ordering of messages. I can see causal telemetry being useful for post-hoc analyses and real time verification of BGP networks. Finding BGP misconfigurations in real time requires precise information on the content and ordering of update messages, exactly the sort of insight causal telemetry gives you into packet histories.

Finally, by leveraging programmable switches, causal telemetry can be used to relate events in the data plane, such as PFC messages, however, it can also be used to log control plane events, such as BGP updates. As long as the messages use the same logical clock, they can be ordered relative to each other.

3 Causal Telemetry

As described in [10], the causal telemetry approach leverages In-band Network Telemetry (INT), a feature that many modern switches support, allowing a packet to log the states it encounters as it traverses the network. However, INT only captures the state of a single packet and falls short when it comes to network wide state. Additionally, capturing logs of multiple packets does not give insight into relating events between switches/packets. The key insight of causal telemetry is to differentiate between 3 kinds of events: *send, receive, and internal*. Using Lamport's happened before relation, corresponding send and receive events can be used to partially order all events in the network [9]. The classic approach to storing this state is in Space-time diagrams. These diagrams can be represented as cyclic-directed-graphs, where events within a node are totally ordered, and related send-receive events between nodes are represented by a directed edge as in Figure 1.

Using INT, only a few pieces of data need to be appended to packets to build Space-time diagrams. Every event has an associated event identifier, which is comprised of a node id (AS number in this case) and a monotonically increasing local logical timestamp. For internal events, the event identifier is simply stored in the log. Send events have the event identifier embedded into the message, and corresponding receive events extract the identifier and record both the sender and receiver in the log. Using this scheme, it is easy to build the Space-time diagrams and order the internal events using the timestamp and order events between devices using send-receive and Lamport's happened before relation.

4 SimpleBGP

Modern BGP routing software suites such as Bird or Quagga are incredibly heavyweight containing tons of functionality and complex routing code. For the purposes of getting a BGP system running with causal telemetry, I only needed a simple BGP routing library with core functionality. Having no success finding anything open source, I built my own SimpleBGP. I implemented it to follow the RFC for

route announcement, backup paths, route selection, a keepalive mechanism for failure detection, and route withdrawal, keeping the implementation lightweight. Thus, it was fairly simple to add the extra telemetry required to build the Space-time diagram.

I used Mininet, a popular open source tool for building complex networks on a commodity laptop. Mininet uses linux namespaces to create virtual nodes and ethernet links between them. By running my BGP routing application on each node, I can obtain full connectivity. The extensive python API allowed me to build a simple interface where I can specify the network topology I want, including local pref values, and with a single script can bring up the entire network, which quickly converges.

Each router in the topology connects to a subnet with a switch and a single host. After convergence, every host can ping every other host, and the packets take the path decided from the BGP control plane.

Every topology consists of a statically configured centralized server, where routers send "postcards" of every event. These "postcards" are packets that contain the logs. The server then processes these logs, relating send and receive events and building out the Space-time graph.

As described in the Causal Telemetry section, there are 3 types of events recorded in the SimpleBGP Space-time diagram. For SimpleBGP, send and receive events are control plane message that correspond to route updates and withdrawals. Internal events could be logged at any granularity. For the purpose of this study, I decided to log route selection and kernel routing table updates, as well as links going down. This is sufficient to get a view of the data-plane. In fact, causal telemetry gets a view of every possible data plane, as long as you can construct a consistent cut. Using this, you can find bugs that could possibly exist, even if no packet took that particular path.

I built a tool that displays a visual representation of the Space-time diagram created at the central server. I can also envision a query language that could be used at larger scale. I discuss this more in future work. Figure 1 shows an example of a Space-time diagram for a simple topology with 3 routers in a line (please note, after I finished the paper I noticed that the next hop is not 0 indexed. Hopefully that doesn't lead to too much confusion). Each node has a string version of the event. For update events it contains the prefix being advertised, the next hop and the as_path. Internal events are blue, receive events are green, and send events are red. At the beginning, each router advertises its prefix and then selects the route that it receives. The middle router then forwards the advertisements to its neighbors. In this case, there are no backup routes as there exists only a single route between each host. The graph already gets large very quickly, even with just 3 routers and 2 links. For the sake of brevity, no more complex diagrams will be shown, but any arbitrary ones can be generated by simply specifying a network topology.

4.1 Debugging

Building fully distributed applications is non-trivial and debugging can feel almost impossible without the right tools. Similar to the story of Linus Torvalds using git as a tool while developing git, I found the Space-time diagrams to be an invaluable tool while debugging my BGP application. This exercise certainly convinced me how powerful the causal view of state is, and how useful it is for understanding

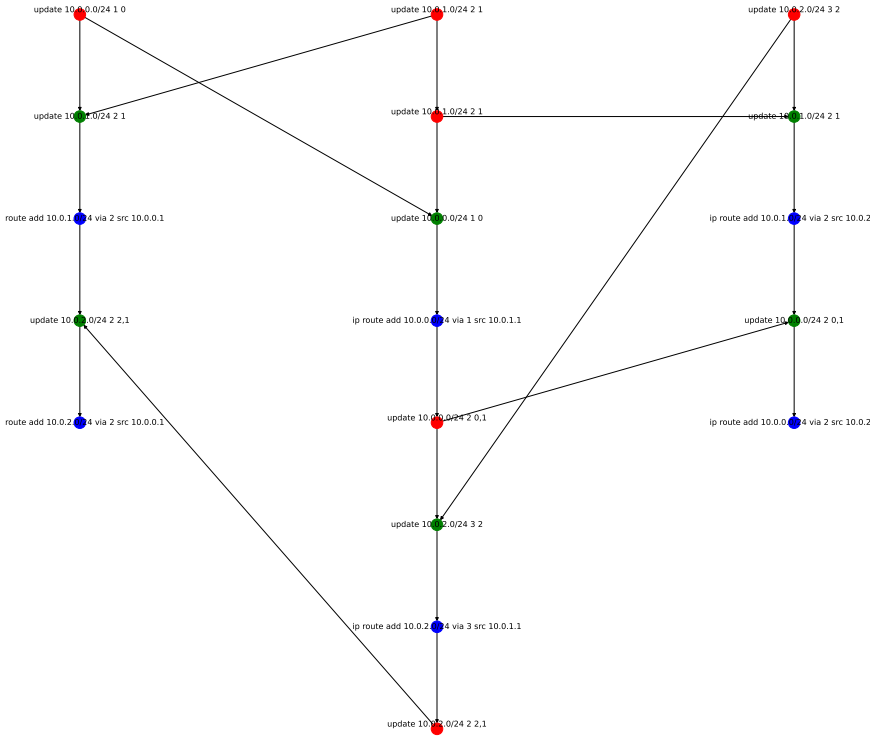

Figure 1: Space-time Diagram 3 Routers

the behavior of your network. Although with a larger topology it is much more complicated, similar to 1, it is possible to follow the application through consistent cuts to get snapshots of state. You can then rebuild the data plane using the logged information from each router. The causal relationship between events allows you to see which update messages correspond to what interval events, and the order that events happen *between* devices in the system. Another really useful approach when viewing the diagrams is to find whatever bad behavior is in the system, caused by some bug or misconfiguration, and then tracing it back in the diagram to see what the root of the issue is, using the causal relationships in the diagram. This allowed me to very quickly figure out what was wrong and fix the bug in the code. Without this view of state that relates events between devices with partial ordering, it would have been much more difficult to debug.

In one particular case, I found that a certain BGP configuration combined with a link going down (which led to a withdrawal of a route), a rather subtle hard to find case, led to a routing loop. By viewing the Space-time diagram and which events led to the loop, I could easily see what the bug was - it had to do with how I was filtering new route update message - and quickly fixed it.

4.2 Example

My first goal was to replicate a known bug and find it using the Space-time diagram. [7] has simple examples of BGP bugs that causal telemetry can easily spot. 2 shows

the bad behavior. Bold lines show the current selected path. If the link between 4 and 5 goes down, 4 will send a route withdrawal to nodes 2 and 3. However, because nodes 2 and 3 know of alternate paths to 4 through each other, they will start to forward data in a transient loop. Typically, this would be resolved quickly with an update message, but the MRAI timer (used to prevent blowup of update messages) could extend this loop to last for greater than 30 seconds.

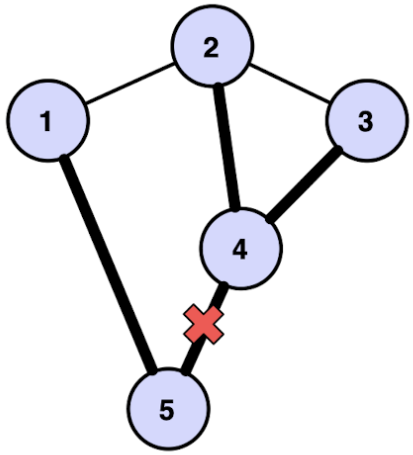

Figure 2: BGP loop caused by link failure

Using SimpleBGP, I was able to replicate this bug and cause a routing loop. As the Space-time diagram is quite large, even at this scale, for the sake of brevity I do not include it. But it was very easy to spot the two internal events that correspond to node 2 and node 3 routing data to node 5 through each other. These events could then easily be traced back to the update message and finally the detection of a link down at node 4. An alternative way to view the Space-time diagram is to see the link down and then traverse the links from there to find the routing loop.

This is just one example of bad BGP behavior that was trivial to spot using causal telemetry. Having a confirmed bug, along with the partially ordered view of global network state shows how powerful causal telemetry can be. I experienced first-hand how useful the Space-time diagram is. Furthermore, it needs very little instrumentation, requiring only a few identifiers to correlate events. I have also begin looking into if causal telemetry could be baked directly into the BGP spec, requiring even less left to be adopted. My idea is to use the community attributes to embed the required data directly into the wire format. Getting this working using commodity switches and open source BGP software would be a big step forward for causaul telemetry, an approach I believe is the right way to do network telemetry and logging.

4.3 Overheads

Each packet in the network and each internal event creates a "postcard" packet which logs the event identifier and any associated information with the event. This is typically less than 100 bytes. While I don't see this becoming an issue when it comes to network congestion, I believe that a bottleneck exists at the centralized

server. While there exist well known techniques for fault tolerance, making the use of a centralized server not an issue in that sense, the sheer amount of data could become an issue. In large data centers and especially in the internet context, there could be millions of update messages, and any given withdraw/update could cascade across the entire network. I explore possible solutions to this in future work.

5 Future Work

While the approach of manually viewing the space-time diagram to find bugs and understand state is useful, especially at smaller scale, I am not satisfied with this as a solution, and certainly, it would never be adopted in this manner.

The next step for future work is a language for querying and searching state space. Most of the exciting future work I see falling into this space. Once the causal telemetry data structure is built and populated there is a lot of rich causal information that can be mined for root causing bugs, understanding network state, and viewing possible dataplanes that could have existed. While queries and searches on data structures such as lists and trees are very well studied and understood, there is less work on directed graphs with cycles. The database community has recently does a lot of work in searches on these types of structures, as well as query languages for expression. I plan on applying this work to the causal telemetry data structure. I foresee the most difficult challenges being how to specify incorrect behavior (or perhaps the correct behavior could be specified), and how to translate that into a search on the graph state space.

The second big area of future work is in pruning the Space-time diagram. Right now, it is infeasible to scale to large datacenters and especially to the internet context. I am interested in exploring pruning techniques for removing unnecessary data. Another approach I want to explore is to optimize to only collect data needed for a given query.

5.1 Learning Outcomes

My learning goals from the beginning of the semester were as follows:

- open source networking projects
- network simulation
  - testing and experimentation
- Low level linux networking framework
- BGP routing and control

At the end of the semester, I can safely say I have achieved each and every one of these. I have also gotten experience with building distributed networked applications. While I think that causal telemetry is the right way to do network telemetry and logging, despite where the research effort may go, creating this system and thinking about these problems has been an invaluable research and learning experience.

6 Related Works

BGP Verification
There has been a lot of work in the area of BGP verification. Feamster and Balakrishnan developed a static checker that takes, as input, BGP configurations and attempts to find two broad classes of faults: Route validity faults where routers may learn routes that do not correspond to usable paths and path visibility faults where routers may fail to learn routes for paths that exist in the network [3]. However, this tool only performs verification at the level of the BGP configuration files and cannot check that routers implement the configuration correctly or that no other routing pathologies occur that interfere with BGP's operations.

Bagpipe is another system that verifies router configurations against a set of BGP policies that are expressed in a declarative language [13]. While Bagpipe is dynamic in the sense of checking router's realtime configurations, it will only check properties that have been written in the language and will fail to catch issues that are not expressed. Additionally, neither of these tools can observe every packet is it flows through the network at runtime. Any error that is missed by the model checker or occurs outside of BGP configurations, such as link outages or switch failures, cannot be checked and will fly under the radar. When (not if) a BGP error does occur, they cannot help an operator figure out what went wrong or localize the issue. Essentially, previous BGP configuration verification tools solve a different problem from causal telemetry and should be used in conjunction.

Networks as Complex Distributed Systems
There has also been recent work on successfully applying well studied distributed systems principles to networking. Synchronized Network Snapshots [14] takes ideas from [2] to build a consistent view of network state. While this is useful, it solves a different problem of seeing network state at a *single point in time*. This view of state cannot look at causal relationships between events over time, that could exist in multiple different states. Additionally, [14] only carries state from when snapshots occur, and could miss important packets necessary for post-hoc root cause analyses.

Consensus Routing [7] addresses the need for consistency in network routing, showing that globally consistent routes, while less responsive, actually increase overall availability of network routes. Consistence Updates [12], similarly aiming for consistency, ensures that any given packet in transit only exists within one network configuration at a time. This ensures that even if the network is in a safe state before and after the update, there is only safe behavior during the transition between states.

Packet Histories and Logs
This work is far from the first to collect packet histories and query logs. However, leveraging switch programmability and in-band telemetry gives a new and powerful way to understand network state as it changes over time. This can be used to root cause misconfigurations and errors or even catch them in real time.

Packet Histories [6] led the way in terms of leveraging SDN and deep programmability to give an operator an easily queryable view of all past events. Causal telemetry builds on this concept, providing a way to order events that happen between different switches (or even endhosts). [8] attempts to find causal relationships between packets

in a log. On the one hand, using the system doesn't require any change to network and router functionality. However, its attempts to use a statistical model to "mine" causation from correlation between co-occurrences of messages. Correlation is not as strong as the happened-before relation, and this approach is likely to miss causal relationships between events that happen infrequently, which is the case for many important network path pathologies such as loops and black holes. Additionally, the preprocessing removes lots of events deemed "unlikely" to be useful for troubleshooting. [8] does not support the powerful querying and root cause analyses that causal telemetry provides on *every relevant event in the network*.

7 Code

The code for the project can be found at:

https://github.com/bennyrubin/CausalTelemetry/tree/main

The code is by no means pretty and was put together to get something working fast. For questions about running it please contact me. There is some work required to get a VM configured and running with mininet.

All the work was done solely by me and there are no ethical concerns with the project.

References

[1] M. Caesar and J. Rexford. "BGP routing policies in ISP networks". In: *IEEE Network* 19.6 (2005), pp. 5–11. DOI: 10.1109/MNET.2005.1541715.

[2] K. Mani Chandy and Leslie Lamport. "Distributed Snapshots: Determining Global States of Distributed Systems". In: *ACM Trans. Comput. Syst.* 3.1 (Feb. 1985), pp. 63–75. ISSN: 0734-2071. DOI: 10.1145/214451.214456. URL: https://doi.org/10.1145/214451.214456.

[3] Nick Feamster and Hari Balakrishnan. "Detecting BGP Configuration Faults with Static Analysis". In: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design amp; Implementation - Volume 2*. NSDI'05. USA: USENIX Association, 2005, pp. 43–56.

[4] Nate Foster et al. "Using Deep Programmability to Put Network Owners in Control". In: *SIGCOMM Comput. Commun. Rev.* 50.4 (Oct. 2020), pp. 82–88. ISSN: 0146-4833. DOI: 10.1145/3431832.3431842. URL: https://doi.org/10.1145/3431832.3431842.

[5] Arpit Gupta et al. "Sonata: Query-Driven Streaming Network Telemetry". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 357–371. ISBN: 9781450355674. DOI: 10.1145/3230543.3230555. URL: https://doi.org/10.1145/3230543.3230555.

[6] Nikhil Handigol et al. "I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks". In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. Seattle, WA: USENIX Association, 2014, pp. 71–85. ISBN: 9781931971096.

[7] John P. John et al. “Consensus Routing: The Internet as a Distributed System”. In: *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*. San Francisco, CA: USENIX Association, Apr. 2008. URL: https://www.usenix.org/conference/nsdi-08/consensus-routing-internet-distributed-system.

[8] Satoru Kobayashi et al. “Mining Causality of Network Events in Log Data”. In: *IEEE Transactions on Network and Service Management* 15 (2018), pp. 53–67.

[9] Leslie Lamport. “Time, Clocks, and the Ordering of Events in a Distributed System”. In: *Commun. ACM* 21.7 (July 1978), pp. 558–565. ISSN: 0001-0782. DOI: 10.1145/359545.359563.

[10] Yunhe Liu, Nate Foster, and Fred B. Schneider. “Causal Network Telemetry”. In: *Proceedings of the 5th International Workshop on P4 in Europe*. EuroP4 ’22. Rome, Italy: Association for Computing Machinery, 2022, pp. 46–52. ISBN: 9781450399357. DOI: 10.1145/3565475.3569084. URL: https://doi.org/10.1145/3565475.3569084.

[11] Ratul Mahajan, David Wetherall, and Tom Anderson. “Understanding BGP Misconfiguration”. In: *SIGCOMM Comput. Commun. Rev.* 32.4 (Aug. 2002), pp. 3–16. ISSN: 0146-4833. DOI: 10.1145/964725.633027. URL: https://doi.org/10.1145/964725.633027.

[12] Mark Reitblatt et al. “Abstractions for Network Update”. In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM ’12. Helsinki, Finland: Association for Computing Machinery, 2012, pp. 323–334. ISBN: 9781450314190. DOI: 10.1145/2342356.2342427. URL: https://doi.org/10.1145/2342356.2342427.

[13] Konstantin Weitz et al. “Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver”. In: *SIGPLAN Not.* 51.10 (Oct. 2016), pp. 765–780. ISSN: 0362-1340. DOI: 10.1145/3022671.2984012. URL: https://doi.org/10.1145/3022671.2984012.

[14] Nofel Yaseen, John Sonchack, and Vincent Liu. “Synchronized Network Snapshots”. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM ’18. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 402–416. ISBN: 9781450355674. DOI: 10.1145/3230543.3230552. URL: https://doi.org/10.1145/3230543.3230552.