Programming language plus simple circuit design could let routers report on their own operation

24 August 2017, by Larry Hardesty

Researchers have come up with a new approach to network monitoring that provides great flexibility in data collection while keeping both the circuit complexity of the router and the number of external analytic servers low. Credit: Massachusetts Institute of Technology

In today's data networks, traffic analysis—determining which links are getting congested and why—is usually done by computers at the network's edge, which try to infer the state of the network from the times at which different data packets reach their destinations.

If the routers inside the network could instead report on their own circumstances, network analysis would be much more precise and efficient, enabling network operators to more rapidly address problems. To that end, router manufacturers have begun equipping their routers with counters that can report on the number of data packets a router has processed in a given time interval.

But raw number counts are only so useful, and giving routers a special-purpose monitoring circuit for every new measurement an operator might want to make isn't practical. The alternative is for routers to ship data packets to outside servers for more complex analysis, but that technique doesn't scale well. A data center with 100,000 servers, for instance, might need another 40,000 to 50,000 servers just to keep up with the flood of router data.

Researchers at MIT, Cisco Systems, and Barefoot Networks have come up with a new approach to network monitoring that provides great flexibility in data collection while keeping both the circuit complexity of the router and the number of external analytic servers low. They describe the work in a paper they're presenting this week at the annual conference of the Association for Computing Machinery's Special Interest Group on Data Communication.

Dubbed Marple, the system consists of a programming language that enables network operators to specify a wide range of network-monitoring tasks and a small set of simple circuit elements that can execute any task specified in the language. Simulations using actual data center traffic statistics suggest that, in the data center setting, Marple should require only one traffic analysis server for every 40 or 50 application servers.

Future-proofing

"There's this big movement toward making routers programmable and making the hardware itself programmable," says Mohammad Alizadeh, the TIBCO Career Development Assistant Professor of Electrical Engineering and Computer Science at MIT and a senior author on the paper. "So we were really motivated to think about what this would mean for network-performance monitoring and measurement. What would I want to be able to

program into the router to make the task of the network operator easier?

"We realized that it's going to be very difficult to try to figure this out by picking out some measurement primitives or algorithms that we know of and saying, here's a module that will allow you to do this, here's a module that will allow you to do that. It would be difficult to get something that's future-proof and general using that approach."

Instead, Alizadeh and his collaborators co-designed the Marple language and the circuitry required to implement Marple queries, with one eye on the expressive flexibility of the language and another on the complexity of the circuits required to realize that flexibility. The team included first author Srinivas Narayana, a postdoc at MIT's Computer Science and Artificial Intelligence Laboratory; Anirudh Sivaraman, Vikram Nathan, and Prateesh Goyal, all MIT graduate students in electrical engineering and computer science; Venkat Arun, an undergraduate at the Indian Institute of Technology Guwahati who visited MIT for a summer; Vimalkumar Jeyakumar of Cisco Tetration Analytics; and Changhoon Kim of Barefoot Networks.

The idea behind Marple is to do as much analysis on the router itself as possible without causing network delays, and then to send the external server summary statistics rather than raw packet data, incurring huge savings in both bandwidth and processing time.

Marple is designed to individually monitor the transmissions of every computer sending data through a router, a number that can easily top 1 million. The problem is that a typical router has enough memory to store statistics on only 64,000 connections or so.

One-way cache

Marple solves this problem through a variation on the common computer science technique of caching, in which frequently used data is stored close to a processing unit for efficient access. Each router has a cache in which it maintains statistics on the data packets it's seen from some fixed number of senders—say, 64,000. If its cache is full, and it receives a packet from yet another sender—the 64,001st—it simply kicks out the data associated with one of the previous 64,000 senders, shipping it off to a support server for storage. If it later receives another packet from the sender it booted, it starts a new cache entry for that sender.

This approach works only if newly booted data can be merged with the data already stored on the server. In the case of packet counting, this is simple enough. If the server records that a given router saw 1,000 packets from sender A, and if the router has seen another 100 packets from sender A since it last emptied A's cache, then at the next update the server simply adds the new 100 packets to the 1,000 it's already recorded.

But the merge process is not so straightforward if the statistic of interest is a weighted average of the number of packets processed per minute or the rate at which packets have been dropped by the network. The researchers' paper, however, includes a theoretical analysis showing that merging is always possible for statistics that are "linear in state."

"Linear" means that any update to the statistic involves multiplying its current value by one number and then adding another number to that product. The "in state" part means that the multiplier and the addend can be the results of mathematical operations performed on some number of previous packet measurements.

"We found that for operations where it wasn't immediately clear how they'd be written in this form, there was always a way to rewrite them into this form," Narayana says. "So it turns out to be a fairly useful class of operations, practically."

"While much work has been done on low-level programmable primitives for measuring performance, these features are impotent without an easier network programming environment so that operators can ask network-level queries without writing low-level queries on multiple routers," says George Varghese, Chancellor's Professor of Computer Science at the University of

California at Los Angeles. "This paper represents an important step toward a programming-language approach to networks, starting with a network programming abstraction. This is in stark contrast to the state of the art today, which is individual router programming, which is fault prone and gives little visibility into the network as a whole. Further, the network programming language is intuitive, using familiar functional-language primitives, reducing the learning curve for operators."

More information: Srinivas Narayana et al. Language-Directed Hardware Design for Network Performance Monitoring, *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17* (2017). DOI: 10.1145/3098822.3098829

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology