Interview with Lixia Zhang and kc claffy

RIK FARROW

Lixia Zhang is a Professor in the Computer Science Department of UCLA. She received her PhD in computer science from MIT, worked at Xerox PARC as a member of the research staff before joining UCLA. She has been leading the Named Data Networking (NDN) project development since 2010. lixia@cs.ucla.edu

kc claffy is founder and director of the Center for Applied Internet Data Analysis (CAIDA), a resident research scientist of the San Diego Supercomputer Center at UC, San Diego, and an Adjunct Professor in the Computer Science and Engineering Department at UC, San Diego. kc@caida.org

Rik Farrow is the editor of *;login*. rik@usenix.org

I first heard about Named Data Networking (NDN) several years ago and just couldn't get excited about it. The very notion of replacing the protocols that underpin the Internet appeared to me to be a Sisyphean task—hopeless, yet time-consuming. Yet today I find myself feeling very differently about NDN.

When I listened to kc claffy's talk during LISA15 [1], I learned three things: NDN researchers have produced working prototypes that solve real problems; NDN secures data itself, instead of securing communication channels as we do today with TLS; and the way we use TCP/IP today is nothing at all like the tasks that protocol was designed for. Let's consider the third point first.

When TCP/IP was developed in the late '70s and early '80s, computers were terribly slow. Networks were proprietary, which meant that only computers from the same vendor could communicate over networks. Computers were also tremendously expensive, making the notion of sharing them across great distances very desirable. The two goals of early sharing were the ability to log in remotely and upload or download files. With the addition of email and netnews, that's exactly what the Internet was used for—until 1994.

Today, over half of Internet traffic is streaming data, with remote login being just a tiny fraction of all traffic. File copying is still common, although a lot of files are copied using BitTorrent clients. As soon as I shared my short introduction with kc claffy and Lixia Zhang, they let me know that I was understating the current state of the Internet.

kc: Yes, and the important bit isn't that it's BitTorrent or streaming: the important bit is that *we don't care exactly where the data comes from, so long as we can verify its provenance and integrity*. So the IP network architecture forces on us something we typically don't need, a point-to-point communications abstraction, and denies us something we typically do need: data integrity/provenance mechanisms.

IP is, without a doubt, not how one would design a network architecture today to serve the current world's communication needs.

With respect to *Sisyphean task*, we recognize the project is no slam dunk, but let's also not forget that in the 1980s and even early 1990s, many people, including those employed by large telecommunications companies in particular, thought that replacing the PSTN (public switched telephone network) network architecture with something as ephemeral and unmanageable as a packet-switching architecture was the sort of pipe dream only academics could afford to pursue. Within 30 years, "impossible" became "inevitable." A thoroughly pessimistic view of this challenge ignores the empirical reality we enjoy every day.

Lixia: Related to the network architecture revolution: one needs to pay attention to the related device revolution. The proliferation of devices in recent years, most of them mobile, from cell phones to cars to billions of sensors, makes address configuration management absolutely intractable. Note that it's the computer revolution that led to packet switching, both using computers to do the switching and to connect computers to each other. A network

architecture has to fit the communication needs of the devices, and applications running on them, that the network connects.

Rik: The people who designed the IP protocols weren't aware of the types of security problems that would face the Internet [6]. When computers were rare, and the network paid for by the Department of Defense, there really wasn't a lot of concern about sharing these resources with people who couldn't be trusted.

NDN researchers have been building applications, on top of protocols, that are both data-centric and security-aware. I must admit that it was the addition of security as a major part of NDN that got my attention. Perhaps the security was there all along and I just missed it. But now, today, we could really use a network that includes scalable security as part of the base protocols.

Lixia: Yes, NDN was designed with essential security building blocks that were neither affordable nor necessary 40 years ago. The research challenge we are pursuing now is making these building blocks easy to use by app developers and end users.

Rik: After reading some of the information found on the NDN site [2], I'd like to know if NDN runs on top of IP and whether the intention is to run NDN alongside IP or to replace IP altogether.

Lixia: Designed as the new narrow waist of the Internet, NDN runs over *anything* that can move packets from one NDN node to another. Not only IP, but WiFi, Bluetooth, Ethernet, IP, UDP/TCP tunnels, or even over tin-can link as shown in this old photo [3].

The current NDN testbed runs software on commodity Linux hosts connected by IP/UDP tunnels, which is exactly how IP started: running over whatever existing communication infrastructure, which was telco wires at the time.

Rik: NDN relies on requests, called *Interests*, which name the data the client is interested in. Can you tell us more about the naming scheme?

Lixia: The easiest analogy is the HTTP request: the URL names the object the browser requests. In a URL, the early part includes the domain name information, followed by application-specific information.

URLs are coded with conventions: for example, by default an HTTP connection uses port 80. NDN namespace structures are conceptually similar, but the naming conventions matter more because applications, as well as the network itself, use names to fetch data. Our work over the last few years has included developing naming conventions to facilitate both application development and automation of signing and verification of data for a few general classes of applications.

Rik: In NDN, returned data is called *Content*, and all Content is signed, which I think is a wonderful idea. But signing relies on having a secure and manageable method of acquiring the public keys of the signing parties. How will NDN deal with this?

kc: That is an excellent and essential question, and sometimes followed by, "You've reduced NDN security to a problem we have utterly failed to solve for any application in TCP/IP: key management and distribution!"

But I think the systems and networking administration community can appreciate more than most that a data-centric security approach can convert hard security problems (e.g., host security) into relatively easier ones (crypto key management). NDN security principal investigator Alex Halderman from the University of Michigan gave a great talk on this last year [4].

Lixia: In a nutshell, securely acquiring the public keys requires one to first establish trust anchors. Today's Internet already has some ways to acquire public keys from trust anchors, i.e., using CAs (certificate agents/authority) to set up secure communication channels. There are new ways to build CAs: for example, DANE (DNS-based Authentication of Named Entities) and Let's Encrypt (https://letsencrypt.org/).

NDN is also developing new approaches to trust anchor establishments, establishing local trust anchors (e.g., the IoT management for all UCLA buildings can configure UCLA rootkey as a trust anchor), and then establishing trusts across local trust anchors. There was early work in this direction by Rivest (Simple Distributed Security Infrastructure (SDSI) [5], but today's IP architecture did not have easily available building blocks to realize it. NDN does.

Rik: The design of IP relies on relatively simple routers performing simple operations—the delivery of a packet to the next hop. NDN seems to require that routers become a lot more intelligent, in that not only must they learn routes and maintain state, they must also intelligently perform caching. Historically, TCP/IP routers have been like switches—hardware designed to pass packets out the correct interface quickly—with relatively slow CPUs to handle routing updates. Will NDN routers require serious processing support, as well as lots of storage?

Lixia: Another very good question. First, today's routers are only simple in textbooks—in the wild, they maintain ridiculous amounts of state and complexity: MPLS for traffic engineering, multicast state for multicast, VPN state for private communications, etc. Why? Because the simple and elegant IP communications model can no longer meet people's needs! For example, delivering CNN news to millions of viewers using point-to-point connections is clearly inefficient, so people want multicast. IP's single best path forwarding can't make good use of today's high-density connectivity, so people want traffic engineering; communication over the public Internet is not secure, so people need VPN; and then there is the eternal promise of QoS.

The complexity that has evolved in response to these needs is not at all elegant. As IP's simple forwarding capabilities were overtaken by the world's needs, people hashed out patch after patch in a reactive and incremental mode. We have ended up not only with many states, but many orthogonal states. Each one does its own job; not only does this not help with other goals, but their interactions may lead to more complex pictures (e.g., MPLS has to design solutions for MPLS multicast!).

Stepping up a level to look at the whole picture: it is not a question of whether data plane needs state, but what is the right data plane state that can address everyone's needs.

The datagram is still the basic unit in packet switched networks. An NDN router keeps just one forwarding state: the Pending Interest Table (PIT). Each entry in PIT records one interest packet that has been forwarded to the next hop(s) and is waiting for a reply. If another request for the same data arrives (a PIT hit), the router simply remembers this new interest's incoming interface so that it'll send a copy of data there when the data arrives. So when the requested data comes back, the router can measure the throughput and RTT in retrieving data; if the data does not come back within the expected RTT, the router can quickly try another neighbor. This router PIT provides per-datagram, per-hop state that gives a network the most flexibility to support a wide variety of functions.

Together with the Content store (cache at each NDN node), the PIT enables:

1. Loop-free, multipath data retrieval
2. Native support of synchronous and asynchronous multicast (i.e., servicing requests from multiple consumers that come at the same time or at different times)
3. Efficient recovery from packet losses (a retransmitted interest finds the data right after the lossy link)
4. Effective flow balancing (i.e., congestion avoidance by regulating how fast to forward Interests to each neighbor node)
5. Real-time recovery from network problems, such as link or node failures (i.e., reducing reliance on slowly converging routing protocols)

People have been trying for years to achieve every one of the above, in separation. NDN uses PIT to get them all in a coherent way.

Rik: Many current applications use a push model, like media streaming or video conferencing, while NDN uses a pull model. The NDN project team has a video conferencing application, ndnrtc, as well as a chat application, ChronoChat. Could you explain how NDN handles these applications using a pull model?

Lixia: The so-called "push model" is only an illusion. There is never any application that uses only one-way packet push (how would the sender know anyone received anything?). All communications are about the receivers, what receivers want (the sender does not gain anything by sending), how well the receivers get it (flow/congestion control). So a receiver has to want some data first. For example, when one wants to join a conference, the receiver sends a request and data will come. That's how all today's conference applications work. Netflix does not stream a movie to you without your request, nor does Hangout push video to your laptop before you click join. NDN does per-packet pulling—that is, one interest pulls one data packet, which leads to the advantages we talked about earlier.

Rik: The NDN project team has also created prototype applications for the Internet of Things. I am guessing that naming could be a great aid in setting up IoT networks. Can you tell us more about how this works?

Lixia: Yes indeed, NDN's use of names in building IoT systems is an ideal fit (as compared to IPv6's way of using addresses to connect IoT devices). Essentially, each IoT device, when installed, will get a name based on where it is installed and what it does, which enables devices to communicate and controllers to send commands to devices using names directly. NDN's built-in security support is also a big plus, as the security solutions for the wired Internet (TLS) really do not fit the IoT environment due to multiple factors: communication overhead, computation complexity, unreliable wireless links, and energy consumption. One writes IoT applications using meaningful names, so names are there already; it's just that today's Internet protocol stack requires a lot of IP-address-related overhead before communication can start.

Rik: I've also heard that there is a lot of interest in NDN by big data communities like meteorology and physics. Why is NDN a better fit for sharing large amounts of data than TCP/IP?

Lixia: The first and foremost factor in data sharing is data naming. If everyone names data in some arbitrary way, it won't be easy for others to figure out what data is available and where; one has to build some lookup tables to describe that file named "foobar" actually contains Los Angeles weather data for January 2016. So even before climate researchers started working with the NDN team, they already recognized their need for a common hierarchical naming structure to facilitate data sharing, so LA weather can be named something like /collectionXX/LA/2016/jan (the name includes other metainfo too).

Second, we are talking about sharing *large* amounts of data, so one certainly wants to fetch data in the most resilient and efficient manner. Resiliency means data fetching can proceed in the face of losses and partial failures. FTP cannot do it because when a TCP connection breaks mid-transfer, everything already fetched is gone (TCP semantics: a connection either successfully closed or aborted). In contrast, NDN names data packets, and every data packet arrived is received; if the current data path

failed, NDN forwarders can quickly and automatically find an alternative path if any exists. Efficiency means fetching data from the nearest location where it is available, multicast delivery with caching—all NDN's built-in functionality.

Third are data security and integrity. Even if big data may not be sensitive to privacy issues, big data users care about data integrity and provenance, and both are ensured by NDN's per data packet crypto signature. Every data packet is assigned a unique name, which ensures that everyone gets the same data if they send requests using the same name.

Rik: Anything else you'd like to say about NDN?

Lixia: We often hear people's concern about any notion of replacing IP: "See how difficult it has been to roll out IPv6, which is only a different version of IP, let alone alone a drastically different architecture."

We want to make very clear that NDN won't *change* the deployed IP infrastructure, v4 or v6, in any way. Rolling out IPv6 has been a challenge precisely because it requires *changes* to the deployed IPv4 infrastructure. NDN, on the other hand, simply makes best use out of IP delivery to move NDN packets, if direct Layer 2 channels are not available. IP got rolled out in the same way over PSTN.

If people say IP replaced PSTN, that is not because IP ever attempted to kick PSTN out, but rather because PSTN went off the stage on its own, as new applications were developed to run over IP. And all legacy applications (e.g., voice call) eventually moved over to IP as well, making IP the global communication infrastructure.

Resources

[1] kc claffy, "A Brief History of a Future Internet: The Named Data Networking Architecture" (slides), USENIX LISA15: http://www.caida.org/publications/presentations/2015/brief_history_future_internet_lisa/brief_history_future_internet_lisa.pdf.

[2] Home site for NDN: http://named-data.net/; FAQ pages: http://named-data.net/project/faq/.

[3] Jon Postel, Steve Crocker, and Vint Cerf portray TCP/IP's ability to be carried by any media, in 1994: http://www.internetsociety.org/what-we-do/grants-and-awards/awards/postel-service-award/photo-gallery.

[4] Alex Halderman, "NDN: A Security Perspective" (slides): http://named-data.net/wp-content/uploads/2015/06/fiapi-2015-security-perspective.pdf.

[5] R. Rivest, B. Lampson, "SDSI—A Simple Distributed Security Infrastructure": https://people.csail.mit.edu/rivest/sdsi10.html.

[6] Craig Timburg, "Net of Insecurity," *Washington Post,* May 30, 2015: http://www.washingtonpost.com/sf/business/2015/05/30/net-of-insecurity-part-1/.