Providing Data to Multiple Users with Reduced Traffic

Garrett Parzych
School of Computing and Augmented Intelligence
Arizona State University
Tempe, USA
gparzych@asu.edu

***Abstract*—I propose a new protocol for sending a single packet to multiple destinations with reduced bandwidth usage. In the current network, a packet destined for multiple locations would have to be sent once for each of the hosts receiving the packet. As these packets travel across the network, it is possible that many of the same routers are used in multiple paths. Using network programmability and the P4 programming language, I devise a protocol that will send the data only once across any link, helping to preserve network bandwidth. I tested my protocol on the FABRIC testbed, and the early results show that the protocol does not incur significant costs in latency.**

I. Introduction and Motivation

Applications where the same data is provided at the same time by a server to many hosts are frequent in computer networks. Some examples include multiplayer video games, livestreamed broadcasts, or video conference calls. In these cases, the data is usually provided to each host individually, filling the network with many copies of the same packet. In cases where these hosts are also situated close to each other in the network, these duplicate packets will likely be sent to the same routers and across the same links for much of their journey. Also, many of these proposed applications will often use the UDP protocol for its increased speed [8], but the lack of congestion control may work to even further increase network traffic.

In this project, I developed a protocol that reduces the resources used in these cases, built using the P4 programming language. Resources are saved by ensuring that links used by several of the server-to-client paths are used only once. The protocol works in two stages: in the first stage, the protocol determines which paths the packet travels on from the server to reach the hosts. In the second stage, the server uses the paths to inform routers where the packet needs to be sent. The server sends a packet containing information on where different paths diverge, and routers are able to read these headers and send packets across multiple links at these places of divergence. I will describe this protocol in more depth in section III.

II. Related Work

Similar ideas to mine have been proposed in the literature, most notably, descriptions of named data networking (NDN) implicitly solve this problem [1]. In NDN, if two hosts request the same data at the same time, a router will take note of this in its memory, forward a single request for the data, then send the data back to both requesters after receiving it. NDN actually solves this problem more generally than I will investigate by caching data at the routers. By doing this, routers are able to provide data to hosts even if they request it at different times. It is this idea that motivated me to investigate if solutions could be created without having to implement NDN. A key difference is that I will not use caching, and thus focus only on providing data to the hosts at the same time.

```
header connection_t {
    bit<32> hopAddr;
    bit<16> nextHdr;
}
```

```
header emission_t {
  bit<32> srcAddr;
  bit<32> dstAddr;
  bit<16> nextHdr;
}
```

Fig. 1. Definition of Connection and Emission headers

A portion of my protocol requires finding the paths followed by packets in the networks. This is a problem for which programmable networks are well-suited, and many solutions using P4 have already been investigated [4], [5]. Many of these solutions involve modifying the packet to record which routers the packets travel on through the network. Once the packet reaches the end host, these headers can be removing and analyzed to see the path through the network. This is the technique that I will employ in my own protocol.

The second part of my protocol involves dictating which links packets should be sent across, an idea similar to source routing. Source routing is a process where a host can choose which path in the network the packet should take. Source routing is another procedure well-suited for programmable networks, and others have investigated using P4 to create implementations [6]

III. Protocol Description

My protocol will work two phases: Connection and Emission. During each phase, a header unique to that phase will be added on top of the Ethernet and IP headers to carry additional information about the network. In the Connection phase, hosts send packets to the server indicating that they would like to receive the information stream from the server. As the packets travel through the network, each router attaches

its IP to the CONNECTION header. This is implemented using a header stack in P4, which allows an array of headers of the same type to be added to a packet. Each individual header will have a field for a router's address, and a field indicating the next header type in in the packet (see figure 1). After receiving the packet, the server looks at the header to see which routers were used in the path. The server will maintain a tree structure rooted at the server that stores the paths through the network to the hosts. This tree will then be used to avoid sending repeat packets during the next phase.

In the EMISSION phase, the server constructs a packet detailing the structure of the tree and sends this to its neighbors. A router in the network uses the tree to forward the packet to all of its children. By doing this, a router will only have to receive the packet once, saving the amount of traffic that would otherwise be used. Although a recursive data structure for a tree seems like the obvious choice, storing and operating on a recursive data structure would be a difficult and possibly impossible task in P4. Instead, the tree is stored using a header stack again. There will be a header in the stack for every node in the tree, containing a field for its IP address, and a field for the address of its parent. There is also a field indicating the next header type in the packet; a full description of the packet can be seen in figure 1. A router will search through the stack and find each node that it is the parent of, then forward the packet to each of its children. In order to help save space, routers will remove headers from the stack that have already been used.

Implementing the EMISSION phase required using packet cloning and recirculating in P4, which allows packets to be duplicated or resent through processing respectively. Since the P4 language does not contain loops, recirculation is how the router is able to iterate through the packet. If the packet does not start with a header matching the router's IP, this first header will be removed and the packet recirculated to try to match on the next header. Once a match is found, a packet is cloned so that one packet can be forwarded while another is recirculated again to find more matches. The packet is constructed so that all of a router's matches will be next to each other in the stack, so when the header no longer matches, the packet is dropped and computation can stop. The rules for cloning and recirculation are somewhat finicky and strict, pseudocode 1 shows how I have managed to implement this behavior. EMISSION packets have been given two new header subtypes in my code, NO_FIRST_MATCH and ALREADY_MATCHED, along with a default value to help recognize what part of the algorithm a packet is in.

Algorithm 1 An algorithm with caption

Ingress:
if IP = emission[0].srcAddr **then**
    Set destination port
    Clone packet (both packets appear at egress)
else if headerType = ALREADY_MATCHED **then**
    Drop packet
else
    headerType ← NO_FIRST_MATCH
    Send packet to egress
end if

Egress:
Pop first element of emission stack
if headerType = NO_FIRST_MATCH **then**
    headerType ← DEFAULT
    Recirculate packet (packet reappears at ingress)
else if Packet is a clone **then**
    headerType ← ALREADY_MATCHED
    Recirculate packet
else
    headerType ← DEFAULT
    Forward packet
end if

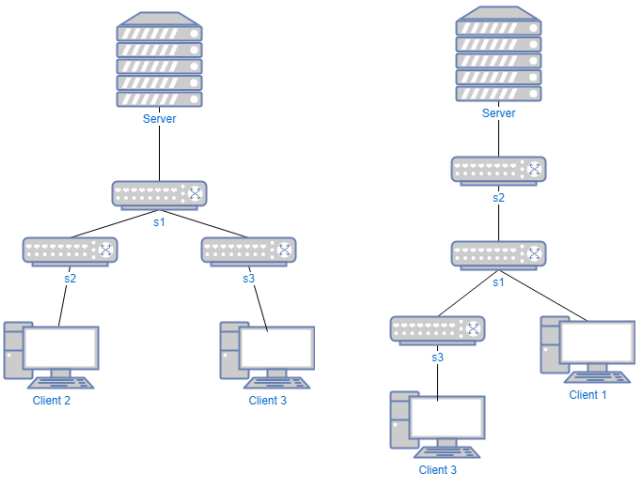

Fig. 2. Topology 1 with host 1 as the server (left) and Topology 2 with host 2 as the server (right)

IV. EXPERIMENTAL SET-UP

After developing my protocol on Mininet, I deployed it on the FABRIC testbed to run more realistic experiments and evaluate its performance. In particular, I was interested in testing the latency of my protocol when compared to conventional forwarding using IP. My proposed applications included video conferences, video games, and live broadcasting, which all typically require lower latencies [7]. For my protocol to be effective, it should therefore be able to run without significantly higher latency.

To run my experiments, I created a topology of three hosts and three routers. I created a python program to implement my server, and deployed it on one of the hosts. Since the topology was asymmetrical, I was able to choose a different host to be the server and run the program on essentially two different topologies. These two topologies can be seen in figure 2.

Tests were run by sending a message from each of the two

clients to the server and waiting for a response. Tcpdump was used to tell the exact times that the messages were sent and received at the client's interface. I tested using both conventional forwarding using IP headers, and forwarding with my own protocol. When testing my protocol, the CONNECTION phase was completed beforehand, so that only the EMISSION packets had their latency measured. I also varied the size of the data in the packets (96, 512, 1024 bytes) to see how this affected the latency. For each combination of topology, protocol, client, and packet size, 20 measurements were taken and the average computed.

V. RESULTS

TABLE I
LATENCY MEASUREMENTS (MILLISECONDS)

Topology and client	**IP Forwarding**			**My Protocol**		
	96 (bytes)	**512**	**1024**	**96**	**512**	**1024**
Topo 1, Client 2	117	116	119	119	119	118
Topo 1, Client 3	144	144	142	144	145	149
Topo 2, Client 1	119	118	116	117	116	116
Topo 2, Client 3	245	246	246	247	246	246

Table I shows the averages for each set of 20 measurements that I took. Interestingly, it does not show a significant increase of latency for my protocol. However, the topologies tested on were small, and so further experimentation would need to be done on larger topologies to see the full effects. Since the size of EMISSION packets grows linearly with the number of nodes in the tree, it is possible that larger networks could begin to incur significant latency costs. There also seems to be some difference between different clients, particularly in the second topology. This is likely a result of Client 3 having three routers in its path to the server, while every other client tested has only two.

VI. SUMMARY, CONCLUSIONS, AND FUTURE WORK

In this paper, I proposed a way to save the number of packets sent in the network for applications where the same data is sent to many hosts at the same time. This protocol could help to reduce the necessary network bandwidth for many applications such as live broadcasts, video conferences, and multiplayer video games. It is made possible by using a programmable network, a paradigm that is seeing rising interest in the research community, and may be the future of network infrastructure.

I hope that my results show that such a protocol could have practical use and provide benefits to the network. However, the extent of my work is still very preliminary and it is not yet clear how useful this protocol could be. Many experiments still need to be completed to have a better understanding of the strengths and weaknesses of my protocol. I would specifically like to conduct experiments on larger topologies in any future research. Another interesting experiment I'd like to try is to create a bottleneck at one of the routers to test how well my protocol preserves bandwidth, if at all.

Future work could also focus on modifying and extending the protocol. For example, improvements could be made for cases where the number of hosts is very high. In these cases, the amount of data stored in the header of EMISSION packets may be very large to account for all these hosts. It would be interesting to come up with ways to keep these header sizes low, perhaps one method could be to maintain a set of many smaller sized trees at the server. Different metrics could be then be used to try to optimize the trees that the protocol creates.

REFERENCES

[1] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named data networking. SIGCOMM Comput. Commun. Rev. 44, 3 (July 2014), 66–73. https://doi.org/10.1145/2656877.2656887

[2] Bosshart, Pat and Daly, Dan and Gibb, Glen and Izzard, Martin and McKeown, Nick and Rexford, Jennifer and Schlesinger, Cole and Talayco, Dan and Vahdat, Amin and Varghese, George and Walker, David. 2014. P4: Programming Protocol-Independent Packet Processors. SIGCOMM Comput. Commun. Rev. 44, 3 (July 2014) 87-95. https://doi.org/10.1145/2656877.2656890

[3] I. Baldin et al., "FABRIC: A National-Scale Programmable Experimental Network Infrastructure," in IEEE Internet Computing, vol. 23, no. 6, pp. 38-47, 1 Nov.-Dec. 2019, doi: 10.1109/MIC.2019.2958545.

[4] J. Hill, M. Aloserij and P. Grosso, "Tracking Network Flows with P4," 2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), Dallas, TX, USA, 2018, pp. 23-32, doi: 10.1109/INDIS.2018.00006.

[5] S. Knossen, J. Hill and P. Grosso, "Hop Recording and Forwarding State Logging: Two Implementations for Path Tracking in P4," 2019 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), Denver, CO, USA, 2019, pp. 36-47, doi: 10.1109/INDIS49552.2019.00010.

[6] C. Dominicini et al., "Deploying PolKA Source Routing in P4 Switches : (Invited Paper)," 2021 International Conference on Optical Network Design and Modeling (ONDM), Gothenburg, Sweden, 2021, pp. 1-3, doi: 10.23919/ONDM51796.2021.9492363.

[7] N. Mohan, L. Corneo, A. Zavodovski, S. Bayhan, W. Wong, and J. Kangasharju, "Pruning edge research with latency shears," Proceedings of the 19th ACM Workshop on Hot Topics in Networks, 2020.

[8] J.-H. Huh, "Reliable user datagram protocol as a solution to latencies in Network Games," MDPI, 02-Nov-2018. [Online]. Available: https://www.mdpi.com/2079-9292/7/11/295. [Accessed: 04-May-2023].