Life in the Fast Lane – viewed from the Confluence Lens

George Varghese
Microsoft Research
Redmond, WA, USA
varghese@microsoft.com

ABSTRACT

The most striking ideas in systems are abstractions such as virtual memory, sockets, or packet scheduling. Algorithmics is the servant of abstraction, allowing the performance of the system to approach that of the underlying hardware. I survey the trajectory of network algorithmics, starting with a focus on speed and scale in the 1990s to measurement and security in the 2000s, using what I call the *confluence lens*.

Confluence sees interdisciplinary work as a merger of two or more disciplines made compelling by an inflection point in the real world, while also producing genuinely transformed ideas. I attempt to show that Network Algorithmics represented a confluence in the 1990s between computer systems, algorithms, and networking. I suggest Confluence Diagrams as a means to identify future interdisciplinary opportunities, and describe the emerging field of Network Verification as a new confluence between networking and programming languages.

1. NETWORK ALGORITHMICS

I use "algorithmics" in a slightly unnatural way to refer to speeding up any good abstraction in computer science that is in danger of being abandoned because of performance. An abstraction is an idealization of reality that allows the user of the abstraction to be more productive by simplifying or idealizing the underling reality.

Great abstractions make life easier for users of the abstraction. For example, relational databases were an advance, but it took years of effort in query planning before relational databases became commonplace.

When I began academic life at Washington University, St. Louis, in the 1990s, the web was exploding. Internet traffic was doubling each year, and so were allocated IP addresses. The only flaw in the ointment was that the beautiful networking abstractions, TCP and IP, were much slower than the raw fiber, which was already reaching 1 Gbps .

TCP, of course, provides the abstraction of two connected queues: this eases the task of data transfer between applications on different machines without considering the details of the underlying network. Similarly, IP offers the simple abstraction of datagram service, sending an isolated message from a source endpoint to a destination endpoint regardless of the variety of link technologies used.

But when the performance wars began in the 1990s with the emergence of fiber, revolutionaries began proposing transports like XTP to replace TCP, and MPLS to replace IP in order to boost performance. This motivated the birth of network algorithmics [30] as a set of techniques to restore the speed of networking abstractions to that of fiber *without* compromising the elegance and ease of use of the abstractions. The rest of this article represents a revisionist history of network algorithmics from the lens of what I call confluences, which I now define.

2. CONFLUENCE DIAGRAMS

The dictionary meaning of a *confluence* is the meeting place of two rivers, as in the Missouri and the Mississippi who meet near St. Louis, or the Tigris and Euphrates who meet near the Garden of Eden. For the purposes of this article, however, a confluence broadly speaking is a meeting place of two *streams of thought*.

We can add some teeth to this definition so it does not reduce merely to interdisciplinary motherhood and apple pie. First, as in the Confluence Diagram shown in Figure 1, distinguish the top stream as the main stream of thought, and the bottom as the *impacting stream*. Second, we seek three distinctives, also depicted in the diagram.

The first requirement is an *inflection point*, a change in the real world, that makes the merger of the streams compelling. Second, the new stream should have a different set of design constraints from its constituent streams, what we might call a *milieu change*, so one must rethink ideas in the merged stream. Thirdly, to ensure that the new stream is a true mixing of streams, there should be evidence of one idea that *transforms* from the impacting stream to the new stream. Mere reuse of existing ideas would be using the impacting stream as a technology – a good thing certainly, but not as exciting as a true confluence of ideas. To make

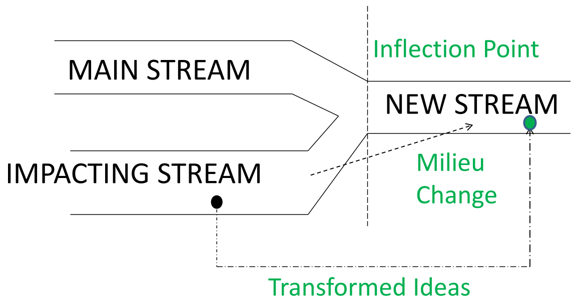

Figure 1: Confluence Diagram: Inflection Point, Milieu Change, and Transformed Ideas

these concepts concrete, consider first an example from painting.

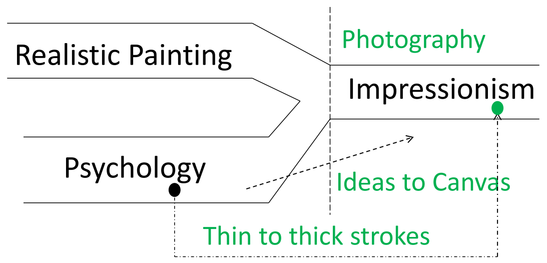

Figure 2: Impressionism as a Confluence

Figure 2 depicts Impressionism as a confluence when mainstream painting in the 1800s was impacted by the emerging field of psychology to form new streams of painting such as impressionism, and later expressionism. The inflection point was the arrival of cheap photography which made painters question the value of merely realistic rendering.

Why not, painters such as Monet may have reasoned, paint the subjective response to a landscape, an impression, something a camera cannot do. This was a milieu change because impressions captured as concepts in psychology now had to be incarnated in paint. There were also transformed ideas: thin, precise brush strokes that delineated borders gave way to blurry thick strokes that mix in the eye at a certain distance.

Why look at existing and new work through the lens of confluence? I will develop this thesis in detail elsewhere. For now, may I suggest that the confluence lens allows us to separate trends from fads by looking for the inflection point; further, the milieu change, once identified, provides a theme for research and a spring of specific research ideas.

The inflection point makes it more likely that the resulting research will have impact, and the milieu change allows creative freedom to rethink ideas in sometimes beautiful ways, balancing the twin desires we have as computer scientists for both beauty and impact. Finally, a discerned confluence can sometimes suggest a new field in the making – a green field area, especially a boon when the original fields (think TCP papers) have matured

3. NETWORK ALGORITHMICS HISTORY

I would like to take a whirlwind tour of various concepts that helped make the Internet fast, but looked at *retrospectively* though the confluence lens.

Fast Servers: My first encounter with algorithmics was when I joined Digital Equipment Corporation and found a beautiful confluence between computer architecture and networking (Figure 3) that led to something that today is called RDMA [2] but was part of the VAX-Cluster system invented by Kronenberg, Strecker and Levy [20]. The inflection point was the realization that one could create cheap clusters of minicomputers; the milieu change was going from a bus in a single computer to many computers connected by a network.

In particular, the inventors of VAXClusters [20] reasoned that since Direct Memory Access was a streamlined way of sending large amounts of data directly from the disk to memory without bothering the CPU, the idea could be extended to Remote DMA from the memory of computer 1 to the memory of computer 2 without the intervention of either CPU. Data was copied only once over the bus, and the overhead of systems calls and interrupts was minimal. Of course, today RDMA is a major force in storage networks, but people forget it was invented in 1986.

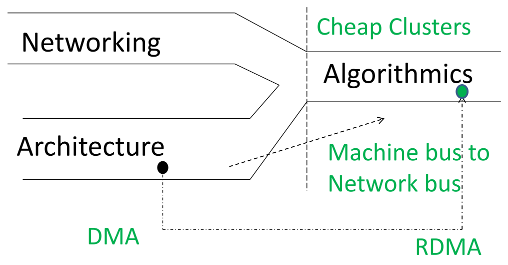

Figure 3: RDMA as a Confluence

Soon after VAXClusters, a new inflection point arose as the Internet began to heat up. Servers were found to be woefully slow because they copied data across layers of software, and because of the overhead of system calls. While RDMA did avoid these overheads, it required protocol changes, and only worked for large data transfers. Thus began a stream of work in SIGCOMM influenced by the Operating System community (the impacting stream), to match the speed gains of RDMA using only local Operating System changes without protocol changes, while retaining structure and protection.

I pick three representative papers. Fbufs from Druschel and Peterson [9] showed it was possible to avoid copies by leveraging page tables. Application Device Channels from Druschel, Peterson and Davy [10] showed how to avoid system calls. Both ideas are alive and well in what people call Zero Copy Interfaces [5] and Virtual Interface Architecture [4]. Finally, Active Messages [32] was roughly concurrent with Application Level Device Channels, and again avoided control overhead by passing information in packets. Beyond ways to streamline data movement, Van Jacobson and Mike Karels showed that TCP performance could be optimized in the expected case using "header prediction" [3] when segments arrive in order. No new transport protocols were needed. The stage had been set for fast servers.

Fast Routers: The first glimmer of a real confluence between algorithms and networks that I encountered arrived because of a new inflection point (Figure 4) around 1996. IPv6 was rumored to be imminent and addresses were now $W = 128$ bits. Simple trie-based schemes were linear in the number of address bits which was too slow. Of course, theoretical computer science had some fast prefix algorithms but they were mostly $O(\log N)$ schemes where N is the number of prefixes, and the milieu was different (Figure 4) because memory accesses and not computation is the dominant metric.

While most theoretical algorithms were content with computing a prefix lookup in milliseconds, an arriving packet had less than a microsecond to be forwarded. It was in puzzling over IPv6 that we discovered a prefix lookup scheme that took $O(\log W)$ memory accesses, which for IP v6 was 7 memory accesses. This to me seemed to be a transformed concept. While $O(\log \log N)$ algorithms were known for lookups [28], they were for exact lookups and more complicated.

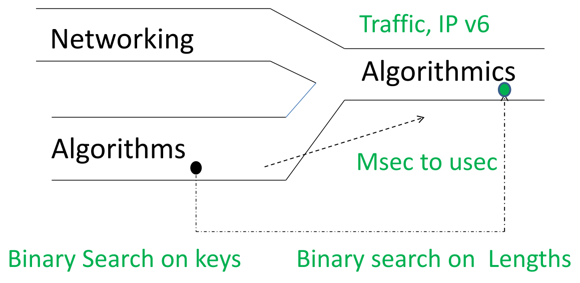

Figure 4: Binary Search on Prefix Lengths as a Confluence.

The story of Binary Search on prefix lengths [33] is a romantic tale of an encounter with two outsiders, and how ideas are "in the air" at a certain time period. I met Jon Turner on the stairs of Bryan Hall in Washington University and he asked why one couldnt do binary search on prefix lengths. Prefixes would first be segregated by their length and then at each length a search for a prefix required only an exact match by hashing.

The usual method starts with the longest length and works backward. Jon, however, wanted to start with the middle prefix length and perform binary search. I thought about his idea for a few minutes and showed him a simple counterexample with two prefixes, one at length 1 and one at length 3. I asked him how one could search in the middle length (2) hash table when there was no prefix of length 2. Some days later, he met me on the same stairs and told me "Easy: for every longer length prefix, add an artificial prefix (called a marker) at all length tables that binary search can reach". Thus in Figure 5, marker 10 is placed in the Length 2 hash table.

This was wonderful as far it went, but there was a flaw. Sometimes markers take you on a wild goose chase to the second half. For example, in Figure 5, when searching for the string 100, marker 10 takes search to the second half towards the entry for 101∗ when the answer (1∗) instead lurks in the first half. Rather than tell Turner about the bug, I decided to fix it myself by *precomputing* the best matching prefix of every marker. For example, the best matching prefixof 10 is precomputed to be 1∗ . If the search process remembers the matching prefix of the last marker encountered, this becomes the answer when search fails without the need for backtracking.

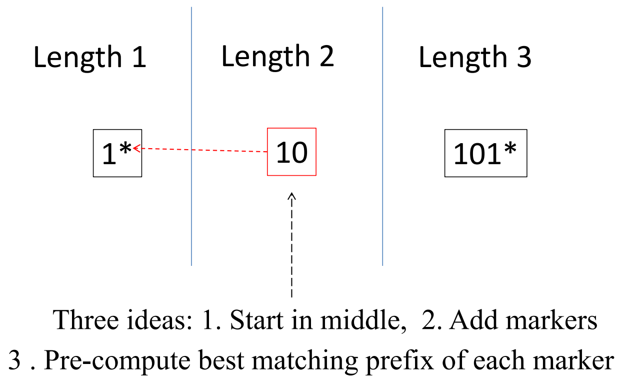
Length 1
Length 2
Length 3
1*
10
101*
Three ideas: 1. Start in middle, 2. Add markers
3 . Pre-compute best matching prefix of each marker

Figure 5: Binary Search on Prefix Lengths. The two prefixes are 1∗ and 101∗. 10 is an artificial entry called a marker used to guide binary search.

Amazingly, on a bus a few days later, I sat next to a really smart Swiss student, Marcel Waldvogel, who was visiting Washington University. He had all the same ideas as Jon and ended with the same bug. So we began working together. Of course, Marcel did all the work, and added a number of elegant ideas of his own such as Rope Search [33].

Next, every packet arriving on an input link of a router is subject to IP lookup to determine its output port and then must be transferred via the guts of the router, often called a switch. Early switches in the 90s, such as the Catalyst 5K from Cisco designed by Tom Edsall, used a simple bus akin to the older PCI bus in a CPU. But as speeds went up, designers realized they had to use a crossbar, which is a set of parallel busses. The simplest technique to schedule a crossbar uses *input queuing*, where packets waiting for an output are placed in a *single* queue at the *input* link.

However, that meant that a packet on an input interface destined to a red output interface could wait (at the input) behind a packet destined to a busy blue output interface, even though the red output interface was free. This is the so-called *Head-of-Line blocking* problem which can reduce throughput by nearly half. This problem resulted in researchers proposing more complex *output queuing* designs. But, as far as I know, output queuing designs never entered the mainstream router market because of their complexity.

A breakthrough occurred, or so it seems to me, around 1992 when Tom Anderson, Chuck Thacker, and others from DEC SRC [7] introduced two new ideas. First, they changed the FIFO interface to one interface at each input for each output, sometimes referred to as VOQs (virtual output queues), as shown in Figure 6. Then information about all non-empty VOQs is sent to a scheduler.

Their second remarkable idea was a maximal matching algorithm called PIM [7] that could be done in hardware in around 5 iterations. One can think of their approach as an Ethernet-like approach per output port. Each output port randomly selects among all input ports that wish to send to it; input ports rejected because of "collisions" retry in the next iteration.

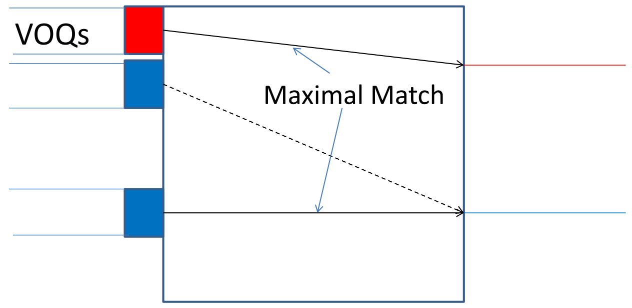

Figure 6: Virtual Output queues and Maximal Matching eliminate Head of Line Blocking

A few years later Nick McKeown introduced iSLIP [24] which can be roughly thought of as a token ring equivalent of the Ethernet-like approach of PIM. The scheduler grants access to each output link based on a rotating priority that cycles through the input ports. While iSLIP can start badly, it generally converges to very good matches after a few iterations. iSLIP was used in the Cisco GSR.

What is compelling here is not just the impact in terms of switch performance but the transformed idea. Maximum match is normally at least linear in the number of inputs in theory. PIM, and then iSLIP, introduced new algorithms for maximal match that computed a match in nanoseconds and worked very well in practice.

While PIM and iSLIP work well for small switches and unicast traffic, Jon Turner showed how to build scalable broadcast switches [27].

Between algorithmics for fast switching, IP Lookups, and packet scheduling, there was generally a sense by the 2000s that people knew how to build fast routers. There was Cisco's Catalyst 6K (the first router with a crossbar), Juniper's M40 (arguably the first router to use ASICs for forwarding), and Cisco's GSR (one of the first commercial routers to avoid head-of-line blocking).

Better still, the solutions scaled with link speeds. Silicon Valley, with a little help from academia, had figured out routers. Was there nothing left for router algorithmics?

4. MEASUREMENT ALGORITHMICS

In the early 2000, a new inflection point arrived for routers. The perils that accompany success beset the Internet in terms of large scale attacks such as worms and Denial-of-Service attacks. Compounding these tactical issues was the strategic difficulty of finding traffic estimates to provision networks. It turns out that both detecting security attacks and finer-grain measurement require detecting patterns *across* packets, a milieu change from standard algorithmics that only detects patterns (such as prefixes) *within* packets.

Naive methods require keeping massive amount of state across packets. However, theoretical bounds show that randomized algorithms can do much better. To illustrate measurement algorithmics as a confluence (Figure 8) between randomized algorithms and algorithmics, consider the following algorithm called Sample-and-Hold, which differs from standard sampling, as employed in say Cisco's NetFlow [1].

Consider the problem of estimating the traffic sent by the heavy "elephant" flow such as $F1$ without keeping track of potentially millions of "mice" flows like $F3$. The basic idea in Sample-and-Hold [11] is that ordinary sampling is used to decide whether a flow like $F1$ is sampled. But once $F1$ is sampled, it is placed in a hash

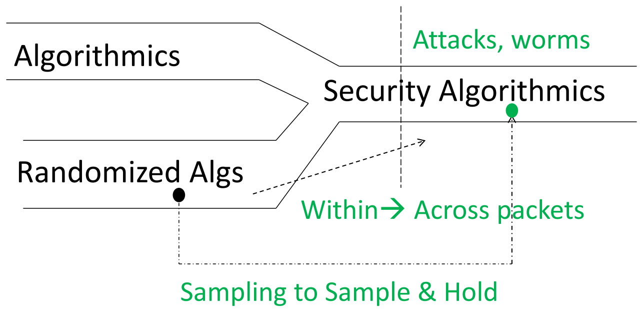

Figure 7: Measurement Algorithmics as a Confluence

table (Flow Table) and *all* subsequent traffic sent by $F1$ is watched.

Unlike sampling, the uncertainty in the measurement of $F1$ occurs only at the start and this translates into a reduction of standard error from $1/\sqrt{M}$ to $1/M$, where M is the amount of memory available for the flow table. M is typically limited, being high speed on-chip memory. Hence, a reduction of error from $1/100$ to $1/10,000$ is appreciable. Of course, the real edge of Sample-and-Hold over sampling occurs because of a milieu change: the router gets to see every packet. By contrast, the Gallup Poll finds it expensive to survey individuals, and hence must resort to standard sampling.

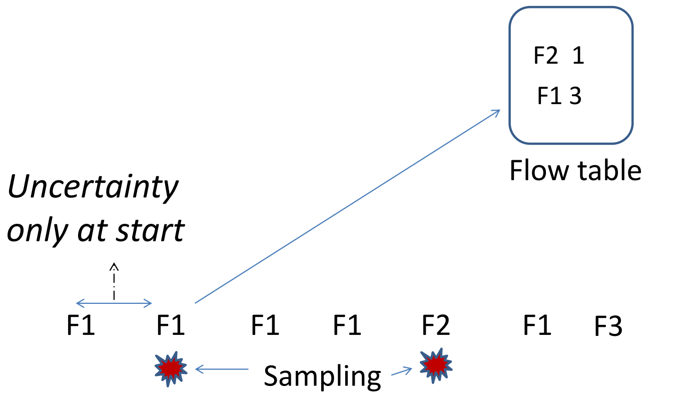

Figure 8: Sample-and-Hold samples flows but then keeps track of all subsequent packets for each sampled flow

Sample-and-Hold was first described in a very nice paper by Gibbons and Mathias on databases [14]. We [11] did, however, add a new analysis and made other changes to fit the networking mileu.

Measurement and security algorithmics have been well developed by many researchers. The following is a sample of three pieces of work I like. First, Elephant traps [16] improve sample-and-hold by periodically removing mice flows that have drifted into the flow table by random chance. Next, researchers have gone beyond heavy-hitter measurements to obtain more complex estimates including flow distributions [21] using small space.

Finally, super-spreader algorithms [31] detect more complex security predicates in streaming fashion such as sources that send packets to a large number of destinations, sometimes indicative of an intruder trying to break into any compromised machine. Some aspects of these ideas are in chips and software; for example, Cisco fabricated a chip based on automated worm detection technology [26]. Despite this, I do not think that measurement algorithmics is mainstream as yet.

5. OTHER NETWORKING CONFLUENCES

Confluences in networking are not new. Examples of past confluences with the impacting field shown in parentheses include queuing networks (Queuing Theory), Pricing the Internet (Economics), and Network Security (Computer Security). More current confluences include Data Center Networks (High Performance Computing) and Wireless Network Coding (Information Theory).

One can argue that each confluence was triggered by an inflection point such as the need to understand packet delays in the early Internet (queuing networks), the shift to the commercial Internet (Internet Pricing), the advent of large scale attacks and cyber-crime (Network Security), the need for large scale data centers to support Cloud Services (Data Center Networks), and the advent of Software Defined Radios together with the dearth of wireless spectrum (Wireless Network Coding).

Each introduces a milieu change from their impacting fields: networks of queues, the reduced importance of marginal costs in Internet economics, the ability to rapidly amplify a security attack, the need to scale clusters to hundreds of thousands of nodes, and the possibility of embracing wireless interference instead of shunning it. Each confluence has produced new ideas including the independence assumption [19], edge pricing [25], ecosystem analysis [22], data center transports with lower latency than TCP [6], and Zig Zag coding to recover information in the face of collisions [15].

An area that excites me personally is what several researchers call "Network Verification". Nick McKeown's SIGCOMM Keynote two years ago, described the *opportunity* in network verification by comparing it to hardware and software verification [23]. Network verification can also be viewed as a *confluence* between Programming Languages and Networking as shown in Figure 9.

The inflection point that makes Network Verification compelling is the emergence of cloud services. Studies [34] have shown that network failures are a significant and debilitating source of operational expenditures that can impact the economic viability of such services.

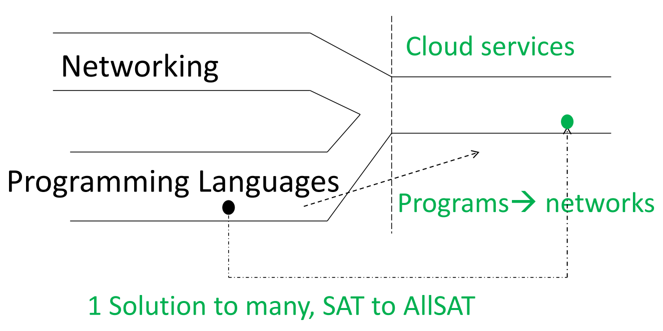

Figure 9: Network Verification as a Confluence

On the other hand, the field of programming language has produced a variety of tools from debuggers to static checkers. Network verification seeks to find analogous tools for networks.

The milieu changes in going from programs to networks. Networks can be regarded as programs that transform packets, and such network "programs" typically have simple control flow. However, the large possible space of packet headers complicates the task compared even to large-scale software such as operating systems. New ideas have emerged in this confluence including new forms of compression to compactly represent the header space [18, 17], the automatic synthesis of forwarding rules at routers [16], the extension of the notion of test coverage to covering links and router queues [34], and the use of causality in network debugging [12].

I started with life in the fast lane and I ended in the 2000s with measurement algorithmics and security algorithmics both of which have been well studied. Are there any unexplored directions for network algorithmics? One I find promising, a confluence between network algorithmics and virtualization as depicted in Figure 10, was brought to my attention by Daniel Firestone and Ramana Kompella.

The milieu change is the placement of network functions in virtual machines running on multiple cores instead of on pipelined router hardware, once again caused by the inflection point of cheap cloud services. Examples of transformed ideas in this space include rethinking TCP performance in virtualized environments [13] and the Route Bricks approach to software router design [8].

6. CONCLUSION

Network algorithmics has played out from a focus on speed and scale in the 90s to a focus on measurement and security algorithmics in the 2000s. A confluence that suggests new problems in network algorithmics may arise from the inflection point caused by network processing in software on virtual machines. Besides current confluences in networking such as Data Center Networking and Wireless Network Coding, Network Verification is a promising new confluence.

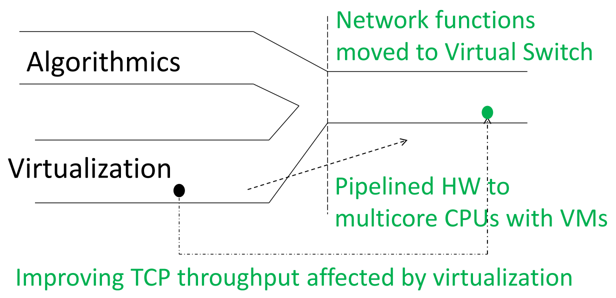
Algorithmics
Network functions moved to Virtual Switch
Virtualization
Pipelined HW to multicore CPUs with VMs
Improving TCP throughput affected by virtualization

Figure 10: Networking using Virtual Machines as a Confluence

Confluence thinking is useful because it allows a researcher to discern a new direction, find a unifying theme, and produce research that balances beauty (via transformed ideas) and impact (via the inflection point). The milieu change allows rethinking ideas in both the existing and impacting fields to produce research ideally of interest to both communities. Of course, this begs the question: are there more systematic techniques to use confluence thinking in research? The slides published online [29] have some hints such as "embrace collisions" and "seek coherence".

This article began with a review of network algorithmics but gradually segued to a framework for thinking about interdisciplinary research. Perhaps, the ultimate excitement is not making things fast but the thrill of discerning a new field with ideas to explore and impact that potentially awaits. I hope seeking confluences will provide readers with that same rush.

7. REFERENCES

[1] Netflow. `http://en.wikipedia.org/wiki/NetFlow`.
[2] Remote direct memory access. `http://en.wikipedia.org/wiki/Remote_direct_memory_access`.
[3] V. Jacobson's notes on TCP header prediction. `http://yangchi.me/v-jacobsons-notes-on-tcp-header-prediction.html`.
[4] Virtual interface architecture. `http://en.wikipedia.org/wiki/Virtual_Interface_Architecture`.
[5] Zero-copy. `http://en.wikipedia.org/wiki/Zero-copy`.
[6] M. Alizadeh and et al. Data center tcp (dctcp). In *Proceedings of ACM SIGCOMM 2010*.
[7] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High-speed switch scheduling for local-area networks. *ACM Trans. Comput. Syst.*, 11(4), Nov. 1993.
[8] M. Dobrescu and et al. Routebricks: Exploiting parallelism to scale software routers. Proceedings of SOSP '09.
[9] P. Druschel and L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. *SIGOPS Oper. Syst. Rev.*, 27(5), Dec. 1993.

[10] P. Druschel, L. Peterson, and B. Davie. Experiences with a high-speed network adaptor: A software perspective. *SIGCOMM Comput. Commun. Rev.*, 24(4), Oct. 1994.

[11] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4), Aug. 2002.

[12] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX NSDI*, 2007.

[13] S. Gamage, R. Kompella, D. Xu, and A. Kangarlou. Protocol responsibility offloading to improve TCP throughput in virtualized environments. *ACM Trans. Comput. Syst.*, 31(3), Aug. 2013.

[14] P. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of SIGMOD 1998*.

[15] S. Gollakota and D. Katabi. Zigzag decoding: Combating hidden terminals in wireless networks. In *Proceedings of ACM SIGCOMM 2008*.

[16] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the "one big switch" abstraction in software-defined networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013.

[17] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[18] A. Khurshid, W. Zhou, M. Caesar, and N. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[19] L. Kleinrock. *Theory, Volume 2, Computer Applications.* Wiley-Interscience, 1975.

[20] N. Kronenberg, H. Levy, and W. Strecker. Vaxcluster: A closely-coupled distributed system. *ACM Trans. Comput. Syst.*, 4(2), May 1986.

[21] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. *SIGMETRICS Perform. Eval. Rev.*

[22] K. Levchenko and et al. Click trajectories: End-to-end analysis of the spam value chain. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy.*

[23] N. McKeown. Mind the gap. `http://yuba.stanford.edu/~nickm/talks/Sigcomm%202012%20POSTED.ppt`.

[24] N. McKeown. iSLIP: A scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.*, 7(2), Apr. 1999.

[25] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in computer networks: Reshaping the research agenda. *SIGCOMM Comput. Commun. Rev.*, 26, 1996.

[26] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, 2004.

[27] J. Turner. Design of a broadcast packet switching network. In *Proceedings of Infocom 1986.*

[28] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, SFCS '75, 1975.

[29] G. Varghese. Life in the fast lane. `http://conferences.sigcomm.org/sigcomm/2014/doc/slides/2.pdf`.

[30] G. Varghese. *Network Algorithmics.* Morgan-Kaufman, 2004.

[31] S. Venkataraman, D. Song, P. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *in Proceedings of 15th IEEE Symposium on High Performance Interconnects*, 2007.

[32] T. von Eicken and et al. Active messages: A mechanism for integrated communication and computation. *SIGARCH Comput. Archit. News*, 20(2), Apr. 1992.

[33] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed ip routing lookups. In *Proceedings of the ACM SIGCOMM '97*, 1997.

[34] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown. Automatic test packet generation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012.