IronStack: performance, stability and security for power grid data networks

Zhiyuan Teo, Vera Kutsenko, Ken Birman, Robbert van Renesse
Department of Computer Science
Cornell University
Email: {zteo, vsk23, ken, rvr} @cs.cornell.edu

1 Abstract

Operators of the nationwide power grid use proprietary data networks to monitor and manage their power distribution systems. These purpose-built, wide area communication networks connect a complex array of equipment ranging from PMUs and synchrophasers to SCADA systems. Collectively, these equipment form part of an intricate feedback system that ensures the stability of the power grid. In support of this mission, the operational requirements of these networks mandates high performance, reliability, and security. We designed IronStack, a system to address these concerns. By using cutting-edge software defined networking technology, IronStack is able to use multiple network paths to improve communications bandwidth and latency, provide seamless failure recovery, and ensure signals security. Additionally, IronStack is incrementally deployable and backward-compatible with existing switching infrastructure.

2 Introduction

Power grid operators have the unique challenge of operating wide area data networks that both drive and depend on power systems. While various systems have been proposed to handle data buffering and processing [1], prior to our effort very little attention has been paid to the operational characteristics of the underlying networks used for data transport. The vulnerability of these data networks to attacks and disruptions – manmade or otherwise – represents a valid concern that needs to be thoroughly addressed.

Present power grid data networks are predominantly run using microwave relays and signal multiplexing on power cables. Although these have proven acceptable over time, the growth of big data in this coming age of smart grid systems means that existing capacity on these data links could be rapidly saturated in the near future. A new and better network technology is needed.

The technology of choice that runs data networks in virtually every other industry is Ethernet. Indeed, Ethernet has enjoyed ubiquity in datacenter and enterprise networking applications for its low cost and ease of use, requiring very little configuration for ordinary operation. Hosts need only plug themselves into Ethernet switches; the switches are then self-organizing and will automatically calculate routes from any source to any destination.

Unfortunately, for the simplicity that Ethernet offers, it also suffers from several severe restrictions, including one that mandates a loop-free topology for correct operation. The ramifications of this simple restriction are that Ethernet networks do not typically feature link redundancy, and where redundancy exists, they cannot be taken advantage of without resorting to delicate and complicated configuration. This is at odds with the plug-and-play vision of Ethernet, where it would intuitively have been expected that additional links introduced between network switching elements should have the effect of automatically and transparently increasing redundancy and performance. In reality, without arduous manual configuration, redundant links are typically left unused until primary failures force them into action. Worse, failure recovery and redundant link activation typically take between several seconds to half a minute. In the context of power grid data networks that convey critical fault data from sensors, this fault recovery time may result in unacceptable knowledge gaps that can severely impede decision making.

Another consequence of the Ethernet design is that it becomes possible for attackers to identify strategic pathways or physical locations where data flow is likely to transit. By infiltrating these locations, attackers can gain access to raw data streams, reading or modifying them at will. It is also conceivable that attackers can masquerade as legitimate sensors and feed malformed data to processing units, with the effect that power management systems may be tricked into taking destabilizing actions. Alternatively, attackers can launch physical or cyber attacks on strategic bottlenecks to cripple the network, severing critical data flow.

Our motivation with the IronStack system is to tease apart these Ethernet problems and craft carefully engineered solutions with software-defined networking techniques. The core contributions of our work include: (1) improvements to end-to-end network performance through packet processing and redundant routing, (2) realization of high-assurance networking through zero-downtime failure recovery and (3) techniques for security through blacklist avoidance, signals obfuscation and transparent encryption, while (4) being fully backward-compatible with existing network infrastructure and equipment.

3 Background

3.1 Deficiencies in the power grid

An increasingly large portion of the national power grid is dependent on data networks for command, communications and control (C3). These data networks frequently carry critical information pertaining to the health of the grid, often through sensor readings, that are then used to make decisions for the next stable operating state of the grid. However, a chicken-and-egg cyclic dependency exists between the two: a data network cannot survive without power; conversely, without data, the grid cannot operate in a safe and stable manner.

Apart from physical infrastructure attacks, one of the weakest links in this delicate balancing act is the data network and the software that depends on it. There is emerging consensus that the power grid has numerous vulnerabilities and is susceptible to large scale remote cyberattacks that can result in real, crippling infrastructural damages. As an example, Stuxnet is a well-known malware that quickly spread through data networks and was directly responsible for the destruction of about 1000 nuclear enrichment centrifuges in Iran. It is conceivable that a similar attack could be launched against power grid hardware in the US, with devastating physical and economic effects.

Another problem in the concurrent use of data networks to support grid operations is the inherent risk of critical data flow disruptions during network equipment outages. Such failures can occur for many reasons, including wear-and-tear, accidents and uncorrelated power losses. Without access to current data, grid operators are at risk for a cascading chain of failures.

3.2 Convergence of big energy and big data

With the emergence of the next generation smart grid, the amount of data that is expected to flow and be processed at control stations will sharply increase. Cisco's surveys [9] have shown that nearly one in four IT managers expect network load to triple over the next two years; the power grid is no exception. In fact, the vision of a smart grid learning, adapting, and controlling the power grid will require big increases in real time data transmission and network load. However, current power grid communications infrastructure uses antiquated technology that will need to be overhauled in order to support such an increase.

Part of the need to support a higher network load comes from the emerging use of synchrophasers. Since 2004, the usage of synchrophasers in the power grid has been increasing. In the grid, synchrophasers use phasor measurement units also known as PMUs to measure real time current, voltage and frequency at distributed locations across the grid. Each of the phasor measurement units timestamps the data that it receives before sending them off to a local SCADA system. Timestamping these measurements allows administrators to have a global view and understanding of the activities on in the grid. Each such device generates 10kb/s or more data, with stringent latency requirements on the links that forward these data to the control centers. As the number of synchrophaser units in the power grid increases, so will real time data and the need for strong and consistent reliability in the network which is difficult to support in current infrastructure.

3.3 Software-defined networking

Software-defined networking (SDN) is a modern abstraction that allows access to a network switch's routing fabric. In SDN models, the switch's control plane is made accessible to a special external software entity (known as a controller), to whom all data switching decisions are delegated. This control plane has complete command of the data forwarding plane, the latter of which is where units of network data (known as packets) are transferred between physical ports on the switch itself. There is also some limited capability to transfer packets between the data forwarding plane and the control plane, a useful feature that we exploit in our system to implement some key functionality.

The most widely deployed SDN standard today is known as OpenFlow. OpenFlow is managed by the Open Networking Foundation and has seen significant evolution through multiple versions. The most recent version of OpenFlow is 1.3, although many switches that are marketed as OpenFlow-capable today support only OpenFlow 1.0. Part of the difficulty lies in the fact that the successive versions of the standard have increased complexity and are not backward-compatible, necessitating support for multiple firmware versions. Our system takes into account the industry momentum at present, and strives to operate on the greatest number of devices by using only features that currently enjoy widespread support.

On the software end, there are multiple efforts to develop operational OpenFlow controllers, each with varying degrees of programmability, complexity and running speed. Some of the more popular and open-source controllers include POX [10] (a generic Python-based system), Floodlight [11] (Java-based) and ovs-controller [12] (a C-based reference controller written by Open vSwitch). For the kinds of network-level services we aimed to provide, we could not find a controller that allowed us fine-grained access to functionality necessary to implement the features presented in this section. For this reason, we wrote our own controller, IronStack, entirely from ground-up in C++ with no support from third-party software libraries.

4 Design

In our effort to modernize power grid data networks with IronStack, we identified three primary objectives: high performance, high assurance and high security. Also important, but not critical, are the pragmatic economic considerations that our solution should be incrementally deployable and preferably fully backward-compatible with existing hardware and software.

4.1 Performance and assurance

IronStack borrows some ideas from RAID [2], a set of redundancy schemes commonly used to protect data by utilizing multiple hard drives. Analogously, the redundancy in data networks are provided by multiple disjoint paths from a source to a destination. However, current networks do not usually feature multiple disjoint paths because they are tedious to design, require a fair degree of manual configuration, and are difficult to maintain in a safe configuration over extended periods of time [4]. Also, software that takes advantage of multiple paths is rare in practice [3]. IronStack solves these problems by automatically generating a safe configuration for any given network topology, while allowing multiple paths to be used simultaneously without the need for any laborious configuration or forethought. IronStack is self-configuring, self-adapting and self-healing, so repeated changes to the network topology do not affect the operation of dependent network software. Thus, IronStack automatically manages the network efficiently in a way that is transparent to users.

The way IronStack uses multiple paths in the network can be seen as a continuum of tradeoffs between latency/reliability and bandwidth efficiency. At one extreme end of the spectrum, each packet in a flow can be replicated onto multiple disjoint paths. The receiving end delivers the first arriving packet to the application and discards the duplicates. Such a scheme minimizes latency and improves the stability of the flow, while also tolerating up to $n-1$ link or switch failures, at a cost of n times the bandwidth.

On the other extreme end of the spectrum, each disjoint path can be seen as a separate channel through which flows can be sent through, so each successive packet in a flow can be sent down whichever path is first available (thus avoiding the problem of sending too many packets down congested paths). In a lightly loaded network, approximately $1/n$ of the packets in a flow can be sent down each path. This scheme maximizes bandwidth efficiency but clearly sacrifices on flow stability and latency, since the entire flow is now dependent on the slowest link. It also does not tolerate link failures although such tolerance may not be necessary if the software protocol can handle it (eg. TCP with selective acknowledgements).

In between these two ends, a k out of n scheme may be used to reap some benefits from both the abovementioned ideas. In this hybridized scheme, individual data bits at the source are striped across n multiple packets, each of which then travels down a different path towards its destination. At the receiving end, only k out of these n striped packets are required for complete assembly of the original data. Hence, this scheme has a latency equivalent to the $n-k+1$th slowest link at any instant, while having a bandwidth efficiency ratio of $\frac{1}{\lceil n/k \rceil}$. This is essentially a form of forward error correction.

4.2 Security

IronStack adopts a defensive perspective on threat modeling. In the IronStack threat model, we assume that an adversary operates on the network and is interested in gaining access to the raw data in a network flow. Where such raw data is protected by encryption, we assume that the adversary is interested in signals intelligence. The adversary can perform a variety of attacks, perhaps by acting as a man-in-the-middle, snooping on sensitive data, modifying data in transit or by passively identifying patterns in communications. We identify three possible security measures for IronStack users. These security mechanisms are orthogonal to the performance and assurance components of IronStack, and can in fact be used simultaneously.

4.3 Localized adversary

When the operating location of the adversary is known, IronStack can blacklist the affected parts of the network and instead construct paths that do not take protected flows through compromised network elements. Since data never transits the adversary's location, it is not possible for the adversary to perform any kind of meaningful attack. Consequently, it is also impossible for the attacker to deduce any signals intelligence from the protected flow.

4.4 Non-localized adversary

If the adversary's location is not known, or if the presence of an adversary is uncertain, blacklisting will not help. However, IronStack can still reduce the problem by randomly distributing data over multiple disjoint paths. This has the effect of obfuscating the signal profile of a flow, making it difficult for an adversary to deduce patterns and ascertain the nature of the flow. Furthermore, any information that the attacker gains is only partial, since the distribution of the data over multiple paths ensures that no single path contains all packets to a flow.

4.5 Transparent end-to-end encryption

Where raw data is streamed unprotected from end-hosts, it is possible for IronStack to intercept these packets and apply encryption to them, such that the resultant flow is resistant to snooping and modification while in transit. The encryption is automatically stripped at the IronStack-controlled switch immediately connected to the end-host, which then expects and sees the data in its raw form. Encryption can be applied in parallel with the abovementioned techniques for avoiding adversaries.

5 Backward compatibility and retrofitting

A practical consideration in the wide-scale deployment of IronStack is the potential cost involved in such an endeavor. While some IronStack functionalities can be supported natively on existing end hosts, others require substantial data packet processing that must be done with special software. During our engineering design process, we examined several options that could enable the use of all IronStack capabilities: retrofitting existing systems with new software, augmenting the IronStack controller with mechanisms to perform packet processing on behalf of the client, and constructing support middleboxes that can handle packet processing on behalf of the client. Table 5.1 summaries the various IronStack designs and their tradeoffs:

	original	retrofit	controller	middlebox
minimal latency	no	yes	maybe	yes
maximal bandwidth	no	yes	maybe	yes
hybrid scheme	no	yes	maybe	yes
blacklisting	no	yes	yes	yes
random paths	no	yes	yes	yes
transparent encryption	no	yes	yes	yes
deployment ease	NA	hard	easy	medium
scalable	NA	yes	no	yes
cost	NA	high	low	medium

Table 5.1: IronStack engineering design tradeoffs.

5.1 Systems retrofitting

A systems retrofit will involve updating the operating systems kernel of each affected host to recognize IronStack packets and process them correctly before delivering data onto their target applications. This is the cleanest and most scalable method of supporting IronStack functionality, since it does not involve the use of computational resources beyond the communicating endpoints. However, such support is contingent on the feasibility and acceptability of modifying such operating systems kernels. For many types of embedded systems – particularly those that run modern grid sensors – or for systems running proprietary operating systems, it is difficult to modify existing kernel code to handle IronStack packet processing. Even where such modification is possible, it may not be acceptable to system administrators, who must now be liable for a larger software attack surface to support IronStack.

5.2 Using the IronStack controller

The IronStack controller is capable of limited data transfers between a switch's control plane and its data plane. It is thus possible for IronStack to entirely handle packet processing on behalf of the end hosts. The advantage of such an approach is that it enables all IronStack capabilities to the end hosts without necessitating any kind of hardware or software change, with the effect that end hosts are completely unaware of an underlying change to the network. In light of the difficulties in modifying individual network equipment operating system kernels as mentioned in the preceding section, such transparency is valuable since changes are localized to the switching equipment and its controller. However, this approach is not scalable because the bandwidth for transfers between the data plane and the control plane is limited. Consequently, it is not possible to service too many concurrent IronStack function requests simultaneously.

5.3 Middleboxes for packet processing

A more scalable approach for IronStack would be to allow the controller to perform some limited functions, but to delegate packet processing duties to an external dedicated entity. This dedicated entity could be a single consumer-grade computer, a cluster of servers on the same network rack, or an array of NetFPGA boards, scaling according to equipment availability the expected processing load generated by network users. This approach maximizes IronStack controller responsiveness and scales very well to high flow counts, however it requires more dedicated equipment and power to operate in aggregate.

6 Implementation status

As at this time of writing, we have completed implementation on a basic prototype of the IronStack controller. This prototype system was recently demonstrated at the MIT Energy Conference and the ARPA-E Energy Summit that were held in February 2014. In our experimental and demonstration setups, the controller ran on a Dell Optiplex 990 with 8 cores and 16Gb of RAM. Our controller handled all required functions and packet processing in their entirety, and interfaced with a Dell S4810 high capacity OpenFlow switch. This switch was selected for its ability to partition into eight virtual switches, from which various network topologies could be explored. Two TCP sensor streams were sent over this network, one with multipath protection transparently provided by IronStack and one without. We were able to show that the protected data flow was immune to network disruptions, while the unprotected flow suffered from stops and starts as the network underwent fault remediation.

7 Related Work

Some work has been accomplished that have relevance to the IronStack system. RAID [2] is the classic work that explores various techniques of storing data on independent disks for the purpose of improving redundancy and performance. Data storage using RAID is largely organized into standardized schemes, with RAID0 corresponding to no redundancy (thus allowing the full utilization of all independent disks), RAID1 corresponding to direct mirroring (simple replication of data across multiple disks) and higher RAID levels corresponding to more complex data striping methods. These ideas have direct counterparts in IronStack where the data is written to disjoint network paths as opposed to independent disks.

SPAIN [4] is an Ethernet-based solution that implements redundancy by mapping strategically computed paths to separate VLANs. Functionally, the objectives of SPAIN and IronStack are similar on the performance and assurance end: they both provide increased bisection bandwidth and resistance to network failures. However, SPAINs implementation relies on static, preinstalled paths, and cannot adapt to substantial network topology changes. Consequently, while robust to individual failures, SPAIN is of limited use in a power grid data network where topology changes due to power or equipment outages are likely. Furthermore, SPAIN does not perform data packet processing and thus cannot feature the continuum of latency/bandwidth tradeoffs that is attainable in IronStack. SPAIN is also a not a security technology by design and does not implement blacklisting, signals-intelligence obfuscation, or transparent end-to-end encryption.

Multipath TCP [5], like IronStack, explores the use of multiple paths to improve overall connection performance. Conceptually, MPTCP takes a stream of data and distributes it across multiple network interfaces, where each network interface would ideally lead to a different connecting path to the destination. It is critical to note that MPTCP works on the L3 network layer and is agnostic to the underlying physical communication paths, so in fact the multiple paths as idealized by the standard could really be tunneling over the same L2 physical layer links. While MPTCP enjoys the cost convenience of not needing any modifications on the existing network, it does require multihoming on devices that wish to take advantage of it. Multihoming may not be possible on many devices that cannot be outfitted with a second network interface card. Furthermore, support for MPTCP is sparse [3] at best, and only caters to the TCP protocol. MPTCP also does not have mechanisms to implement blacklisting. IronStack solves all of these problems; it does not have any of MPTCP's drawbacks since it operates on the L2 layer, and can physically ensure that path diversity or path constraints are satisfied.

SDN-based solutions for robust networking have also been examined. FatTire [6] is a programming language that allows users to specify network redundancy levels, as well as the specific paths that their data packets should transit in a network. The program is then efficiently compiled down to OpenFlow rules that get installed on network switches. This approach naturally facilitates blacklisting and implementing seamless network link failovers. However, it requires substantial domain-specific knowledge to operate and write in the language. Also, while FatTire allows for failover redundancy, it can neither boost aggregational bandwidth nor perform actions to improve data security.

Hedera [8] is an example of a dynamic flow scheduler that actively schedules multi-stage switching fabrics in order to improve bisectional bandwidth. It works by collecting flow information from all constituent network switches and maintaining a global view of the network in order to intelligently re-route traffic around bottlenecks. Again, as with most preceding work, Hedera does not process packets and is thus unable to implement any of the packet striping schemes. It also does not have mechanisms to blacklist network switches or obfuscate signals intelligence of network flows.

8 Future work

In the future, we would like to make IronStack still easier to use by completely removing the need to specify redundancy and bandwidth parameters. The system will instead use machine learning techniques to automatically tune the existing flows subject to restrictions imposed by security policies. It is anticipated that this change can attain much higher operating efficiencies than manual tuning. We also plan to augment IronStack with TCP-R [7], a capability orthogonal to our network-level assurance that provides software-level fault tolerance. This vastly increases the robustness of applications, since they would be protected from both hardware and software failures. On the scalability frontier, we would like to build custom hardware accelerators that will function as plug-and-play cards to replace software packet processing in the IronStack controller. This will greatly improve the service capacity of our system. Finally, we also plan to write software modules that will provide applications a considerable degree of automatic network-level protection against malformed or dangerous data.

9 Conclusion

In this paper we presented IronStack, a novel OpenFlow switch controller that provides high performance, high assurance and high security guarantees for power grid data networks. IronStack is incrementally deployable, necessitating minimal upgrade investment costs beyond the gradual transition to OpenFlow-capable hardware. It is also backward-compatible with existing hardware and software, requiring little configuration and maintenance. Finally, IronStack is also scalable and can thus be used in diverse networking scenarios. Our prototype system featuring a controller-only implementation was recently demonstrated at the MIT Energy Conference and the ARPA-E Energy Summit, and showed that our techniques were sound and practical. We believe that IronStack represents a fundamental engineering advancement in data networks for the power grid, and can be an important tool in the grand scheme of modernizing the power industry.

10 Acknowledgements

This work is supported by grants from DARPA, NSF and DoE/ARPA-E.

11 References

[1] Gjermundrod, Harald, et al. "GridStat: A flexible QoS-managed data dissemination framework for the power grid." IEEE Transactions on Power Delivery 24.1 (2009): 136.
[2] Patterson, David A., Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). Vol. 17. No. 3. ACM, 1988.
[3] Kostopoulos, Alexandros, et al. "Towards multipath TCP adoption: challenges and opportunities." Next Generation Internet (NGI), 2010 6th EURO-NF Conference on. IEEE, 2010.
[4] Mudigonda, Jayaram, et al., SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies. NSDI. 2010.
[5] Ford, Alan, et al. TCP Extensions for Multipath Operation with Multiple Addresses: draft-ietf-mptcp-multiaddressed-03. No. Internet draft (draft-ietf-mptcp-multiaddressed-07). Roke Manor, 2011.
[6] Reitblatt, Mark, et al. "FatTire: declarative fault tolerance for software-defined networks." Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013.
[7] Surton, Robert, et al. http://www.cs.cornell.edu/ burgess/tcpr/
[8] Al-Fares, Mohammad, et al. "Hedera: Dynamic Flow Scheduling for Data Center Networks." NSDI. Vol. 10. 2010.
[9] Summary: Top 10 Global Survey Results, Cisco Connected World Technology Report on Big Data. http://www.cisco.com/c/dam/en/us/solutions/enterprise/connected world-technology-report/Top-10-Survey-Results-CCWTR-Big-Data.pdf
[10] POX. http://www.noxrepo.org/pox/about-pox/
[11] Floodlight OpenFlow controller. http://www.projectfloodlight.o
[12] Open vSwitch. http://openvswitch.org/