Peer-to-Peer Secure Updates for Heterogeneous Edge Devices

Herry Herry*, Emily Band†, Colin Perkins‡, and Jeremy Singer*
School of Computing Science, University of Glasgow, Glasgow G12 8QQ, United Kingdom
*{herry.herry, jeremy.singer}@glasgow.ac.uk, †2038561B@student.gla.ac.uk, ‡csp@csperkins.org

***Abstract*—We consider the problem of securely distributing software updates to large scale clusters of heterogeneous edge compute nodes. Such nodes are needed to support the Internet of Things and low-latency edge compute scenarios, but are difficult to manage and update because they exist at the edge of the network behind NATs and firewalls that limit connectivity, or because they are mobile and have intermittent network access. We present a prototype secure update architecture for these devices that uses the combination of peer-to-peer protocols and automated NAT traversal techniques. This demonstrates that edge devices can be managed in an environment subject to partial or intermittent network connectivity, where there is not necessarily direct access from a management node to the devices being updated.**

I. Introduction

Deployment of smart campuses and smart cities use low-cost, low-power edge compute devices to build sensor and control systems, and to provide smart edge compute nodes. Such systems often start small, with a few tens of nodes, but deployments can rapidly scale to many thousands of devices, and future systems can be expected to be still larger scale. The devices that comprise these systems can be in inaccessible locations, be mobile, or be in private residences. Remote administration is essential to ensure that security updates are applied, and to install new applications and services.

Standard cluster management tools, such as Puppet [1], Chef [2], and Ansible [3], have a track record of allowing administrators to successfully manage large numbers of devices. However, these tools rely on either direct push network access from the central management systems to the devices being managed, or on having the devices have access to a well-connected central server from which they can pull updates. Such cluster management tools work well for always-on nodes in well-connected networks, such as data centres, but are not suitable for more the distributed, mobile, ad-hoc, and otherwise poorly-connected environments we consider.

Additionally, relying on a central management server to provide updates, whether push from that server to edge nodes or pulling updates from the server, introduces robustness and scalability concerns. The central server is a single point of failure, vulnerable to hardware and software failures and malicious attacks, and will likely become a performance bottleneck as the system grows. The use of replicated servers, content distribution networks, and similar mechanisms can address these concerns, but come with significant financial and complexity costs.

978-1-5386-3416-5/18/$31.00 © 2018 IEEE

In this paper, we consider an alternative to such centralised systems, and explore whether peer-to-peer overlay protocols can enable secure, scalable, and cost-effective system updates for large-scale networks of heterogeneous edge devices. We consider how to build an overlay in networks with limited connectivity, such as those behind network address translator (NAT) devices, restricted firewalls, or with intermittent connections to the Internet, and how to use this to deploy updates in a scaleable manner.

To support this, we have designed and built a prototype decentralised management framework that distributes system updates and management scripts via a peer-to-peer overlay. This allows us to manage devices that do not have direct network access such as nodes behind network address translators (NATs) or firewalls. Also, the nodes in local network could share the same update files, saving the bandwidth to the external network. Moreover, we may not need a dedicated update server since every node can provide the updates for the others. Our framework combines several key techniques:

1) STUN-based UDP hole punching to discover and open NAT bindings;
2) a gossip protocol to deliver short messages, similar in scope to Serf[4], to distribute update notifications; and
3) BitTorrent to securely distribute the software updates.

Our contribution is to demonstrate that clusters of edge devices can be managed in an environment subject to partial or intermittent network connectivity, and in the presence of NATs and firewalls, where there is not necessarily direct access from a management node to the devices being updated.

The remainder of this paper is organized as follows. We begin, in Section II, but outlining the challenges inherent in building a peer-to-peer update protocol that can operate in today's challenged network environment. We describe our proposed system architecture in section III, and give an example of its operation in in Section IV. Related work is discussed in section V. Finally, section VI concludes and considers future work.

II. Challenges in Peer-to-Peer Cluster Updates

The deployment and subsequent need to manage large-scale clusters of heterogeneous edge devices forces us to consider issues that do not occur in traditional data centre networks. These devices lack a standard device hardware configuration;

tend to be relatively underpowered, and may not be able to run heavy-weight management tools due to performance or power constraints. In addition, they tend to have only limited network access due to the presence of firewalls, NATs, or intermittent connectivity, and so cannot be assumed to be directly accessible by a management system.

The issue of limited network connectivity is a significant challenge for the management and secure updating infrastructure. Traditional data centres are built around the assumption that hosts being managed are either directly accessible to the management system, or they can directly access the management system, and that the network is generally reliable. This is increasingly not be the case for the devices we consider. There are several reasons for this:

1) devices will be installed in independently operated residential or commercial networks at the edge of the Internet, and so will be subject to the security policies of those networks, and will be protected by the firewalls enforcing those policies.
2) devices located in independently operated networks may be in different addressing realms, and traffic may need to pass through a NAT between the device being managed and the management system.
3) devices may not always be connected to the Internet, perhaps because they are mobile, power constrained, or otherwise have limited access.

Traditional cluster management tools fail in these environments since they do not account for NAT devices and partial connectivity, and frequently do not consider intermittent connectivity. Management and updating tools need to evolve to allow node management via indirect, peer-to-peer, connections that traverse firewalls that prohibit direct connection to the system being managed. They must also incorporate automatic NAT traversal, building on protocols such as STUN and ICE [5], [6] to allow NAT hole-punching and node access without manual NAT configuration[1].

Existing cluster management tools scale by assuming a restricted, largely homogeneous, network environment, but this is not scenario we envisage for sensor and control networks, or edge computing. Such systems will increasingly be deployed in the wild, at the edge of the network, where the network performance and configuration is not predictable. Update and management tools must become smart about maintaining connectivity in the face of these difficulties to manage nodes to which there's no direct access. In the remainder of this paper, we outline a prototype system that aims to begin to address these challenges.

III. System Architecture

Our peer-to-peer secure update framework requires every node to have the following capabilities:

[1]Existing peer-to-peer management tools, for example APT-P2P [7] and HashiCorp Serf [4], require NAT/firewall pinholes to be manually configured, to allow access by the management tools, limiting their applicability to environments with system administration support.

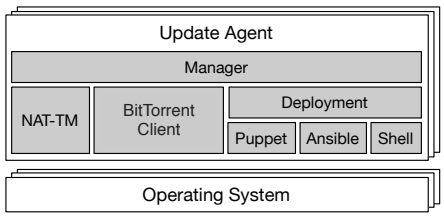

Fig. 1: The architecture of Peer-to-Peer Secure Update framework.

1) it can receive update notifications and download the update file from at least one other peer;
2) it can relay notifications/share updates with other peers;
3) both (1) and (2) should work even though the peer is behind NAT or firewall;
4) it can independently deploy updates on local node; and
5) it can verify integrity of notifications and updates to avoid malicious activity, e.g., man-in-the-middle attacks.

Driven by the above requirements, we design a framework whose architecture is depicted in figure 1. The framework uses an agent, that runs on every node, which consists of four components: manager, NAT traversal messaging (NAT-TM), BitTorrent client, and deployment. The design is mainly targetting a system that runs a Linux operation system, although it can be adapted to other platforms.

The *manager* is a component that receives an update notification in the form of torrent-file either from an administrator or other agents. It uses a public-key, which is installed on every node, in order to verify the integrity of torrent-file. It triggers the download by passing the torrent-file to BitTorrent client, and calls *deployment* component to apply the update once the download has finished. It also relays the notification to its peers using a peer-to-peer overlay network provided by *NAT-TM*.

The *NAT Traversal Messaging* component (NAT-TM) provides a peer-to-peer overlay network which uses a gossip protocol to distribute a message to its peer, in scope similar to the one that is implemented in Serf [4]. It uses the UDP hole punching via the STUN protocol [8], [5] to enable bi-directional communication with nodes behind NATs.

The *BitTorrent client* receives a torrent-file from *manager* and uses it to download the update file from its peers. Our prototype is using an off-the-shelf BitTorrent client i.e. Transmission [9] which supports Distributed Hash Table (DHT) [10] that enables trackerless network. *Manager* monitors the download status from the log file generated by Transmission.

Finally, the *deployment* component receives the update file and the target resource identifier from *manager*. It then invokes a provisioning tool such as Puppet or just simply a shell script to apply the update.

The standard format of torrent-file contains trackers' address and files' information, where each file's information consists of filename, file-length, piece-length, and cryptographic-hash of file pieces [11]. However, there are some important information that are missing from this standard, which are

required by our framework to work properly, including: 1) digital signature, for integrity checking; 2) resource identifier, which distinguishes two different target resources that will be updated; and 3) version, that helps the agent to ignore outdated update which might still exist in the network. Because of this, we enrich the standard format with these new information. For signature, we implement a draft scheme that is proposed in [12] which uses a private-key owned by the administrator, and a public-key installed on every node, for signing and verifying the torrent-file. Unfortunately, we are not aware of any proposal that provides the other two. Thus, we add a Uniform Resource Identifier (URI) and a (monotonically increasing) version into our new format, which is called as `torrent-file++`.

Our current design considers two main threats: 1) man-in-the-middle attack, and 2) re-distribution of an old, potentially vulnerable, version of the software. The agent mitigates the former by discarding any `torrent-file++` that fail the digital signature verification. For the latter, the agent uses the URI to retreive the last update version (from its local database) that had been applied before, and only accepts the `torrent-file++` with newer version.

We choose BitTorrent because it has been widely proven to be reliable to securely distributing files. The file integrity can be verified using cryptographic-hash of file-pieces. Popular BitTorrent clients, such as Transmission, implement NAT Port Mapping protocol (NAT-PMP) [13] to work with NATs. We also can limit the network usage of the BitTorrent clients for low bandwidth environments.

IV. Example

This section gives an example that illustrates the operations of our peer-to-peer secure update framework.

Assume we have a 4-nodes system as illustrated in figure 2a, where: node 1 is fully connected to the internet; node 2 is in a private network connected to the internet via a NAT; and node 3 and 4 are in a private network connected to the internet via a restricted firewall which does not allow any communication initiated from outside. Each node runs an agent whose architecture is depicted in figure 1.

As administrator, we would like to apply an update on all nodes to fix a security vulnerability. The process of deploying this update using our peer-to-peer framework can be summarised as follows:

- (Figure 2a) The administrator creates a torrent-file from the update-file, signs it using a private-key, and uploads both files into node 1. Note that the files could be uploaded to any node.
- (Figure 2b) The agent of node 1 verifies the integrity of torrent-file using its public-key as well as the integrity of update-file using cryptographic hashes of file piece available in the torrent-file. The update will be deleted if the verification fails. Otherwise, the agent deploys it on local node and then passes the files to a BitTorrent client for sharing with other nodes.
- (Figure 2b) The agent of node 2 performs a UDP hole punching by sending a message to the STUN server through the NAT. This opens a particular port (EPort2) on the NAT router which is accessible from internet. The STUN server records this information by keeping the internal IP and port (IP2, Port2) and the external IP and port (EIP2, EPort2) of node 2 in its session table. Since the router will close EPort2 if there is no activity for a period of time, then the agent periodically sends a keepalive message to the STUN server to keep the port open.
- (Figure 2b) For node 3 and 4, the administrator should manually reconfigure the firewall to open a particular port allowing incoming UDP packets to a trusted node, which in this case is node 3. While node 4 will use node 3 as a bridge. The agent of node 3 sends a message to the STUN server that allows its internal IP/port (IP3, Port3) and external IP/port (EIP3, EPort3) to be recorded in the session table.
- (Figure 2c) The STUN server sends its session table to node 1, 2 and 3 so that each node knows which IP/port that should be used to communicate with the others. This activity is done periodically in order to propagate any changes on the session table whenever some nodes joins or leaves the cluster.
- (Figure 2d) After the agent of node 1 knows the external IP/port of node 2 and 3, then it sends the torrent-file to these nodes as a notification that a new update is available. The agents of node 2 and 3 then verify the integrity of torrent-file using their public-keys. If it is verified, then the agent of node 3 sends the torrent-file to node 4, which will also verify it using its public-key.
- (Figure 2e) Since the agents of node 2, 3, and 4 already have the torrent-file, they could start downloading the update-file by submitting the torrent-file to the BitTorrent client.
- (Figure 2f) When the download has finished, the agents can deploy the update on their local node by invoking necessary management scripts.

V. Related Work

Mainstream OS update services feature peer-to-peer content distribution. For instance, the Windows 10 'Delivery Optimization' system [14] provides a peer-based distributed cache, as of late 2016, to mitigate bandwidth issues for large updates [15]. The Debian package management tool has an integrated peer-to-peer download facility called apt-p2p [7], [16]. It fetches update files using BitTorrent, via a transparent http proxy. Package validation requires cryptographic hash value comparisons with known values from a trusted server.

Dale and Liu [16] identify problems with peer-to-peer updating, such as long wait times and different users requiring different subsets of available applications. Zhang et al. [17] present optimizations to the underlying Kademlia distributed hash table (DHT) algorithm to improve the file-piece search

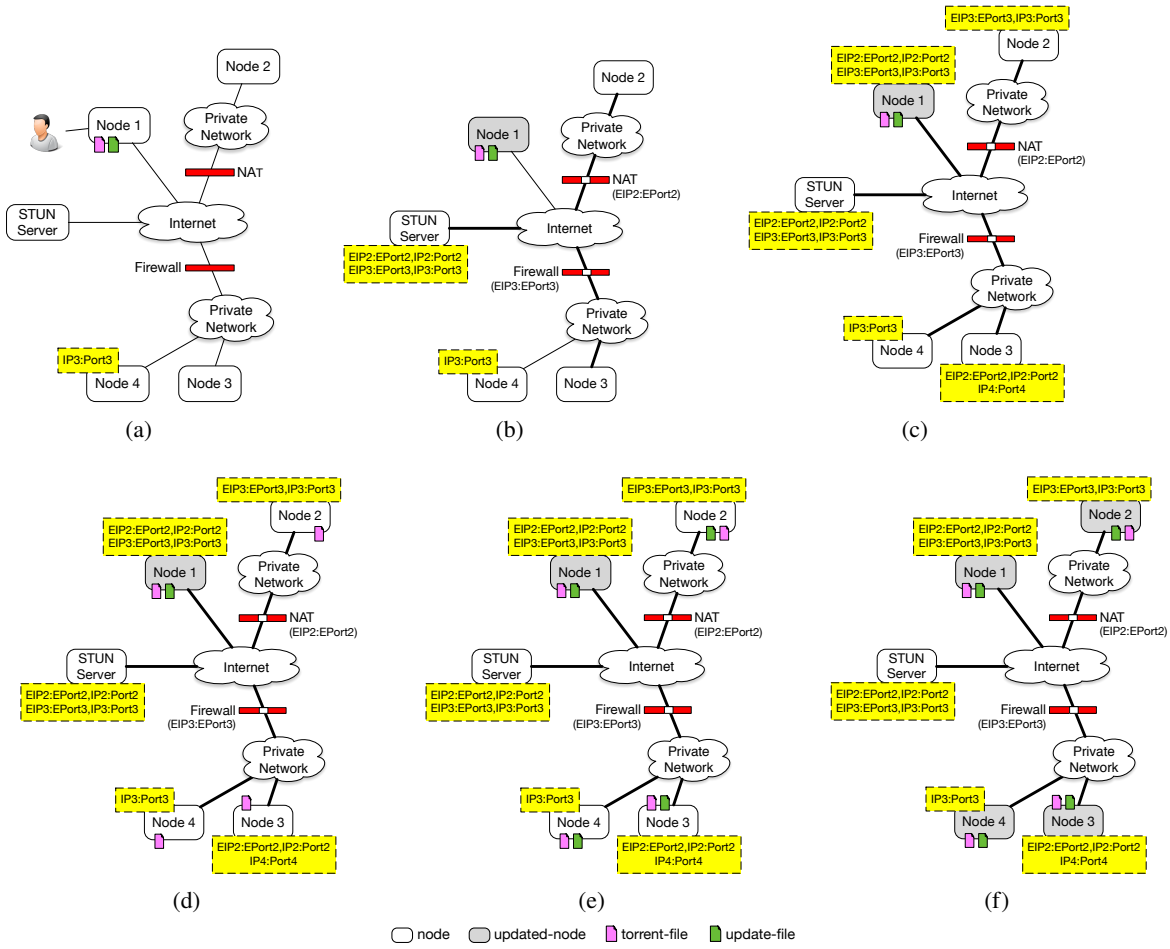

Fig. 2: An example update process of a system where some nodes are behind a NAT (node 2) or a firewall (node 3 and 4). IP_x, Port_x, EIP_x and EPort_x are the internal IP address, the internal port, the external IP address, and the external port of the agent on node_x, respectively.

performance. However these problems are not relevant to the challenges mentioned in section II.

Other system-level peer-to-peer distribution services include peer-npm [18] which handles JavaScript package management. This is a modular system, so it can interface with multiple distributed file system backends to retrieve packages and metadata.

BitTorrent, torrent-tracker, and distributed hash table (DHT) implementations are all well-known aspects of peer-to-peer distribution networks. Our system relies on these openly available technologies.

Serf [4] is a gossip-based protocol [19] that enables decentralized service discovery and orchestration in a cluster of nodes. It supports an event-based system, effectively enabling a peer-to-peer publish/subscribe model. Our system uses a similar protocol to inform IoT nodes of configuration change events.

Puppet [1], Chef [2], and Ansible [3] are industry-standard configuration management tools, provisioning software and services on remote nodes. These tools generally require continuous network connectivity for master/slave style orchestration. Our work involves moving configuration management to a more heterogeneous, distributed network, while still maintaining some compatibility with existing solutions. For instance, we support Puppet updates initiated via our overlay network.

One early suggestion for updating a network of sensor nodes involves assigning a random slot time for each node to download the update from a central server [20].

VI. Conclusion and Future Works

We have developed a peer-to-peer secure update framework and proved that its prototype can update securely the system with partial network connectivity, and works in the presence of NATs or firewalls.

Future work will integrate this peer-to-peer secure update framework into the FRμIT[2] testbed to enhance scalability, and allow us to manage hosts in challenged network environments.

[2]The Federated RaspberryPi μ-Infrastructure Testbed (https://fruit-testbed.org) is a testbed of federated, geo-distributed micro-datacenters. The project aims to aggregate low-cost, low-power, commodity infrastructure to form an efficient and effective compute fabric for key distributed applications.

Acknowledgements

This work is funded by UK Engineering and Physical Sciences Research Council under grant EP/P004024/1.

References

[1] V. Hendrix, D. Benjamin, and Y. Yao, "Scientific Cluster Deployment and Recovery - Using Puppet to simplify cluster management," *Journal of Physics: Conference Series*, vol. 396, no. 4, Dec. 2012.

[2] M. Marschall, *Chef Infrastructure Automation Cookbook*. Packt Publishing, 2013.

[3] M. Mohaan and R. Raithatha, *Learning Ansible : use Ansible to configure your systems, deploy software, and orchestrate advanced IT tasks*. Packt Publishing, 2014.

[4] HashiCorp, "Serf: Decentralized cluster membership, failure detection, and orchestration," https://www.serf.io/, 2017, Accessed: 2017-12-20.

[5] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session traversal utilities for NAT (STUN)," IETF, October 2008, RFC 5389.

[6] J. Rosenberg, "ICE: A protocol for NAT traversal for offer/answer protocols," IETF, April 2010, RFC 5245.

[7] C. Dale, "apt-p2p – apt helper for peer-to-peer downloads of debian packages," http://manpages.ubuntu.com/manpages/zesty/man8/apt-p2p.8.html, 2017, Accessed: 2017-12-20.

[8] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators." in *Proc. USENIX Annual Technical Conference*, 2005.

[9] "Transmission," https://transmissionbt.com, Accessed: 2018-01-25.

[10] A. Loewenstern and A. Norberg, "DHT Protocol," http://www.bittorrent.org/beps/bep_0005.html, January 2008.

[11] B. Cohen, "The BitTorrent protocol specification," http://www.bittorrent.org/beps/bep_0003.html, January 2008.

[12] C. Brown, "Torrent Signing," http://www.bittorrent.org/beps/bep_0035.html, July 2012.

[13] S. Cheshire and M. Krochmal, "NAT Port Mapping Protocol (NAT-PMP)," IETF, April 2013, RFC 6886.

[14] Microsoft, "Windows update delivery optimization and privacy," https://privacy.microsoft.com/en-US/windows-10-windows-update-delivery-optimization, 2017, Accessed: 2018-01-27.

[15] D. Halfin *et al.*, "Deploy updates using Windows Update for Business," https://docs.microsoft.com/en-gb/windows/deployment/update/waas-manage-updates-wufb, 2017.

[16] C. Dale and J. Liu, "apt-p2p: A peer-to-peer distribution system for software package releases and updates," in *IEEE INFOCOM 2009*, 2009, pp. 864–872.

[17] Q. Zhang, J. Yu, L. Luo, J. Ma, Q. Wu, and S. Li, "An optimized dht for linux package distribution," in *15th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2016, pp. 298–305.

[18] S. Whitmore, "peer-npm," https://www.npmjs.com/package/peer-npm, 2017, accessed: 2018-01-27.

[19] P. T. Eugster, R. Guerraoui, A. M. Kermarrec, and L. Massoulie, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[20] G. Pollock, D. Thompson, J. Sventek, and P. Goldsack, "The asymptotic configuration of application components in a distributed system," University of Glasgow, Technical Report, 1998. [Online]. Available: http://eprints.gla.ac.uk/79048/