International Journal for Research in Engineering Application & Management (IJREAM)
ISSN : 2454-9150 Vol-03, Issue 03, Apr 2017

# Automatic Test Packet Generation

[1]Prof. Shinde Vishal, [2]Mr. More Ganesh L, [3]Mr. Lawande Vaibhav D

[1]Asst. Professor, [2,3]UG Student, [1,2,3]Computer Engg. Dept. Shivajirao S.Jondhle College of Engineering & Technology, Asangaon, Maharshatra, India

[2]*ganeshmore30031994@gmail.com*, [3]*vaibhavlawande125@gmail.com*

**Abstract—Networks are getting greater and more compound, yet superintendents rely on elementary tools such as and to debug problems. Thus, an programmed and organised approach for testing and debugging networks called "Automatic Test Packet Generation" (ATPG) is proposed. ATPG reads router configurations and produces a device independent model. The model is used to generate a minimum set of test packets to minimally exercise every link in the network or maximally exercise every rule in the network. Test packets are sent sporadically, and detected failures generate a separate mechanism to localize the fault. ATPG can detect both functional and performance problems. ATPG complements but goes beyond earlier work in static checking or fault localization which only localize faults given liveness results. ATPG protocol is used for implementation and results on two real-world data sets: Stanford University's backbone network and Internet2. A small number of test packets serves to test all rules in these networks: For example, 3500 packets can cover all rules in Stanford backbone network, while 55 are enough to cover all links. Sending 3500 test packets 9 times per second consumes less than 1% of link capacity. ATPG code and the datasets are publicly available.**

**Keyword: Data plane analysis, network troubleshooting, test packet generation.**

## I. INTRODUCTION

It is disreputably hard to rectify networks. Every day, network engineers wrestle with router misconfigurations, fiber cuts, faulty interfaces, mislabeled cables, code bugs, intermittent links, and a several alternative reasons that cause networks to decline fully. Network engineers search out bugs using the foremost elementary to trace down root causes employing a combination of accumulated information and instinct. Debugging networks is simply turning more durable as networks have gotten larger and have gotten a lot of difficult. It's a little surprise that network engineers are labeled "masters of complexity". Think about following example. Suppose a router with a faulty contour card starts sinking packets taciturnly. Alex, who administers ninety nine routers, receives a price ticket from many unfortunate users complaintive regarding connectivity. First, Alex examine router to check if the configuration was modified recently and concludes that the configuration was unhurt. Next, Alex uses her data of the topology to triangulate the faulty device with ping and eventually, she calls a colleague to exchange the line Card. Organizations will modify ATPG to fulfil their requirements; for instance, they'll value more highly to merely check for network aliveness or check each rule to verify security policy.

ATPG may be custom-made to examine just for reachability or for routine also. ATPG will adapt to restraints like demanding test packets from solely a rare places within the network or using uncommon ATPG may also to tuned to allot additional test packets to use additional vital rules. for instance, additional test packets to firewall to Firewall rules to confirm HIPPA compliance.

The results are inspiring . the amount of test packets wanted is surprisingly tiny. For the Stanford network with over 755 000 rules and over ninety VLANs, we have a tendency to solely need 3700 packets to review all forwarding rules and ACLs.

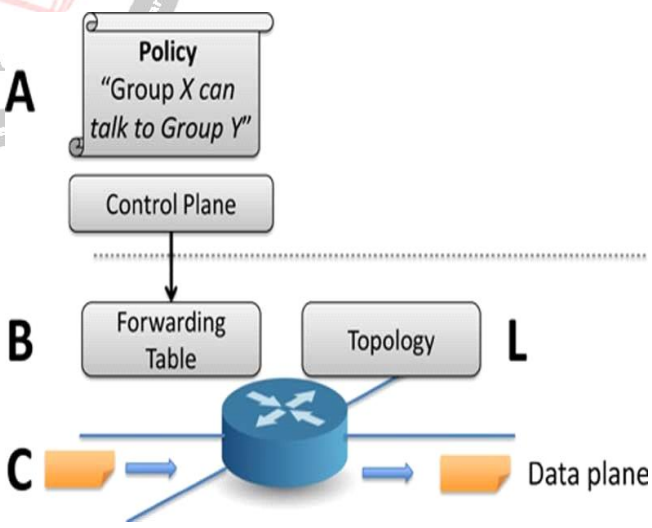



**Fig 1: Static versus Dynamic checking**

On Internet2, 34000 packets serve to study all IPv4 forwarding rules. Put another way, we can crisscross every rule in every router on the Stanford backbone 9 times every second by sending test packets that ingest less than 1% of network bandwidth. The link cover for Stanford is even smaller, around 45 packets, which allows positive live ness testing every millisecond using 1% of network bandwidth.

## II. LITERATURE SURVEY

### *1. KLEE*

It introduce a new symbolic execution tool, KLEE, skilled of automatically generating tests that achieve great coverage on a various set of complex and environmentally intensive programs. This method used KLEE to systematically check all 88 standalone programs in the GNU COREUTILS usefulness suite, which form the core user level environment installed on millions of UNIX systems, and perhaps are the single most profoundly tested set of open source programs in presence. KLEE generated tests achieve high line coverage on average over 89% per tool and significantly shattered the coverage of the developers' own handwritten test suites.

It also used KLEE as a bug finding tool, applying it to 440 applications, where it found 50 solemn bugs, counting three in COREUTILS that had been unexploited for over 15 years. Finally, Is used KLEE to cross-check supposedly identical BUSY BOX and COREUTILS values, finding functional correctness errors and a numerous of irregularities.

### *2. NETWORK TOMOGRAPHY OF BINARY NETWORK PERFORMANCE*

In network performance tomography, characteristics of the network peripheral, such as link loss and packet latency, are concrete from allied end to end measurements. Most work to date is based on manipulating packet level correlations, e.g. of multicast packets or unicast competitions of them. However, these methods are often limited in scope multicast is not broadly deployed or require deployment of additional hardware or software infrastructure. Some recent work has been successful in reaching a less detailed goal: identifying the lossiest network links using only uncorrelated end-to-end measurements. Moreover, the algorithm is sufficiently simple that can analyze its performance explicitly.

### *3. NETWORK TOMOGRAPHY OF BINARY NETWORK PERFORMANCE.*

In network performance tomography, characteristics of the network interior, such as link loss and packet latency, are inferred from correlated end-to-end measurements. Most work to date is based on exploiting packet level correlations, e.g., of multicast packets or unicast emulations of them. However, these methods are often limited in scope-multicast is not widely deployed-or require deployment of additional hardware or software infrastructure. Some recent work has been successful in reaching a less detailed goal: identifying the lossiest network links using only uncorrelated end-to-end measurements. In this approach,

Its abstract the properties of network performance that allow this to be done and exploit them with a quick and simple inference algorithm that, with high likelihood, identifies the worst performing links. It give several examples of real network performance measures that exhibit the required properties. Moreover, the algorithm is sufficiently simple that can analyze its performance explicitly.

1: **input:** Topology *T* ; End-to-end measurements *fXkgk2R*;
2: *Y*0 = 1;
3: *W* = ;;
4: recurse(1);
5: **output:** *W*;
6: do
7: **subroutine** recurse(*k*) *{*
9: **if**(*k 2 R*) *{Yk= Xk}*;
10: **else** *{*
11: *Yk*= max*j2d*(*k*) *Yj*;
12: **foreach**(*j 2 d*(*k*))
*13.}*

## III. EXISTING SYSTEM

Testing aliveness of a network is a fundamental problem for ISPs and huge data centre workers. Sending enquiries between every couple of bound ports is neither comprehensive nor ascendable. It serves to find a marginal set of end to end packets that negotiate each link. However, doing this needs a way of theorizing crosswise device specific configuration files, producing headers and the links they reach, finally determining a least set of test packets To check imposing reliability between policy and the configuration.

**DISADVANTAGES OF EXISTING SYSTEM**

- Not planned to identify aliveness failures, bugs router hardware or software, or enactment problems.
- The two utmost mutual causes of network failure are hardware failures and software bugs, and that problems evident themselves both as approachable failures and throughput/latency dilapidation.
- It check only Network layer status, Not Correct identify the issues in Network.

## IV. PROBLEM STATEMENT

In current system, the administrator manually decides which ping packet to be sent. Sending programs among every pair off boundary ports is neither broad nor scalable. This system is adequate to find least set of end to end packets that travel each link. However, doing this need a way of abstracting across device specific configuration files generating headers and links they reach and finally calculating a minimum set of test packets. It is not designed to identify failures caused from failed links and routers, bugs caused from faulty router hardware or software, and performance problems. The common causes of network failure are hardware failures and software bugs, in which that problems manifest both as reachability failures and throughput/latency degradation. To overcome this we are proposing new system.

TABLE 1. Comparative Study

| Sr. No. | Paper Title | Author's Name | Problem | Solution |
|---|---|---|---|---|
| 1 | Network performance anomaly detection and localization | P. Barford, N. Duffield, A. Ron, and J. Sommers | Location of perform-ance anomalies (e.g., high jitter or loss events) is critical to ensuring the effective operation of network infrastructures. | Framework for detecting and localizing performance anomalies based on using an active probe-enabled measurement infrastructure deployed on the periphery of a network |
| 2 | Robust monitoring of link delays and faults in IP networks | Y. Bejerano and R. Rastogi | Link delays and faults in a Service Provider or Enterprise IP network | Two-phased approach attempts to minimize both the monitoring infrastructure costs as well as the additional traffic due to probe messages. |
| 3 | Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs | C. Cadar, D. Dunbar, and D. Engler | Its time consuming n complex procedure to user | Implement a tool KLEE, capable of automatically generating tests that achieve high coverage on a diverse set of complex |
| 4 | Network tomography of binary network performance characteristics | N. Duffield | Network problems such as link loss and packet latency | network performance that allow this to be done and exploit them with a quick and simple inference |
| 5 | A NICE way to test Open Flow applications | M. Canini, D.Venzano, P. Peresini, D.Kostic | Risk of programming errors that make communication less reliable | Implement a single controller program manages the network, seems to reduce the likelihood of bugs. |

## V. PROPOSED SYSTEM

***ALGORITHM1-TESTPACKET GENERATION***

Main goal is to generate a set of test packets to exercise every rule in every switch function, so that any fault will be observed by at least one test packet. When generating test packets, there are two main con- straint0s:

(1) Port: ATPG must use only test terminals that are available;

(2) Header: ATPG must use only headers that each test terminal is permitted to send.

**Step 1: (Test Packet Selection)**

For a network with the switch functions,$\{T_1,\ldots T_N\}$ , and topology function(T), find the minimum set of test packets to exercise all reachable rules, subject to the port and header constraints. Hence, choose test packets using an algorithm call Test Packet Selection (TPS) Algorithm. TPS first find all the equivalent classes between each pair of available ports.

**TPS Algorithm**

GOOGLE-QUERY-DELAY (tcp_segments)

- S ← tcp_segments[1]
- p ← s
- c ← tcp_segments[2]
- while c ≠ NIL
- do if c.snd_time>p.snd_time and
- c.ack_time>p.ack_time
- then return ( s.snd_time, c.snd_time )
- else
- If
- c.size< MSS
- then return ( s.snd_time, c.snd_time )
- p ← c
- c ← c.next

**Step 2: Generate an all-pairs reachability table**

First start by determining the complete set of packet headers that can be sent from each test terminal, to every other test terminal.

On every terminal port, Apply an all- x header (a header which has all wildcarded bits) to the transfer function of the first hub boxes connected to test terminals.

**Table no.7.1.1 All-pairs reachability table: all possible headers from every terminal to every other terminal, along with the rules**

| Header | Ingress Port | Engress Port | Rule History |
|---|---|---|---|
| h1 | p11 | p12 | {r11, r12, |
| h2 | p21 | p22 | ….} |
| …… | …… | …… | {r21, r22, |
| Hn | pn1 | pn2 | ….} |
| | | | ……. |
| | | | {rn1, rn2, |
| | | | ….} |

**Step 3: Sampling**

Next, It need to pick at least one test packet to exercise every rule. In fact, by picking one packet in an equivalence class, It can test all of the rules reached by the class. The simplest scheme is to randomly pick one packet per class. This scheme only catches faults for which all packets covered by the same rule will experience the same fault. At the other extreme, It want to catch a fault for a specific header within a equivalence class, then it need to test every header in that class.

**Step 4: Compression.**

Several of the test packets picked in Step 2 exercise the same rule. Therefore it will find the minimum subset of the test packets for which the union of their rule histories covers all rules in the network. The cover can be changed to cover all links or all router queues.

***ALGORITHM 2- FAULT LOCALIZATION AND FAULT PROPAGATION***

This algorithm use for marking faulty rules assumes that a test packet will accomplished only if it accomplished at every hop. For perception, consider ping - a ping will succeed only when all the forwarding rules along the path behave correctly. Similarly, if a chain is jammed, any packets that travel through it will acquire higher latency and may fail an end-to-end test.

**Assumption 1 (Fault propagation)**

R(pk) = 1 if and only if □r Є pk:history, R(r; pk) = 1

To mark a faulty rule, we start by finding the least set of potentially faulty rules.

Formally:

**Problem 2 (Fault Localization).**

Given a list of (pk0, R(pk0), (pk1,R(pk1), …tuples, find all r that satisfies ∃pki, R(pki;, r) = 0.

- function NETWORK (packets, switches, Γ)
- for pk0 ϵ packets do
- T←FIND_SWITCH (pk0.p, switches)
- for pk1 ϵ T (pk0) do
- if pk1.p ϵ EdgePorts then
- #Reached edge
- RECORD (pk)
- else
- #Find next hop
- NETWORK (Γ(pk1), switches, Γ)

Hence solve this problem opportunistically and in steps.

**Step 1**: Consider the results from our regular test packets. For every passing test, place all the rules they exercise into the set of passing rules, P. If it similarly define all rules traversed by failing test packets F, then one or more of the rules in F are in error. Therefore F - P is a set of suspect rules.

**Step 2:** ATPG trims to make the set of suspect rules as small as possible by weeding out all the correctly working rules.

For this it make use of the reserved packets (which were the packets eliminated by the Min-Set-Cover). From the reserved packets, we _nd those whose rule history contains exactly one rule from the suspect set and send them. If the test packet fails, it shows that the exercised rule is for sure in error. If it passes, it can remove that rule from the suspect set. Then repeat the same process for the rest of the suspect set.

**Step 3:** In most cases it have a small enough suspect set that can stop here and report them all. However, It can further narrow down the suspect set by sending test packets that exercise two or more of the rules in the suspect set using Step 2's technique. If these test packets pass, it shows that none of the exercised rules are in error and it can remove them from the suspect set. If our Fault Propagation assumption holds, the method will not miss any faults, and therefore will have no false negatives.

## VI. MATHMATICAL MODEL

Specifying models are,
b =0|1|x where,
b= bit
pk = (p, h) where
pk= packet
h =[b0, b1,b2,……,bL] where,
h=header
p = port
T: pk pk
T= Transfer function
Complexity of finding test packet is,
O(TDR^2) and
Complexity of computing reachability from one input port is,
O(DR^2) where,
T= numer of test terminals
D= network diameter
R= average number of rules
Fault model
R(r, pk)= 0, if pk fails at rule r
1, if pk succeed at rule r,
Where, R= result of function
r = rule
end-to-endvesion of result function
R(pk) = 0, if pk fails
1, if pk succeeds
Fault prapogation
R(pk)= 1 if and only if
r Є pk

# VII. SYSTEM ARCHITECTURE

## 1. TEST PACKET GENERATION

Let us consider that the sending and receiving of test packets is occurring in between a bunch of test terminals. Generating least number of "test packets" to execute all the packet processing rules in every node is the main aim of the proposed work. The system must respect two key constraints,

**while generating the test packets.**

1) Port - The tool should only make use of available test terminals,

2) Header - The tool uses the headers for the purpose of granting send permission to every testing terminal.

### Test Packet Selection

The switching functions T1,..Tn, and network topology function Γ, the Test Packet Selection (TPS) algorithm can be used for selecting the test packets to study all packet processing rules which can be reachable, subject to the header and port restraints in a network. In between each and every couple of available nodes, Test Packet Selection algorithm (TPS) is used for finding the equivalent set of classes.

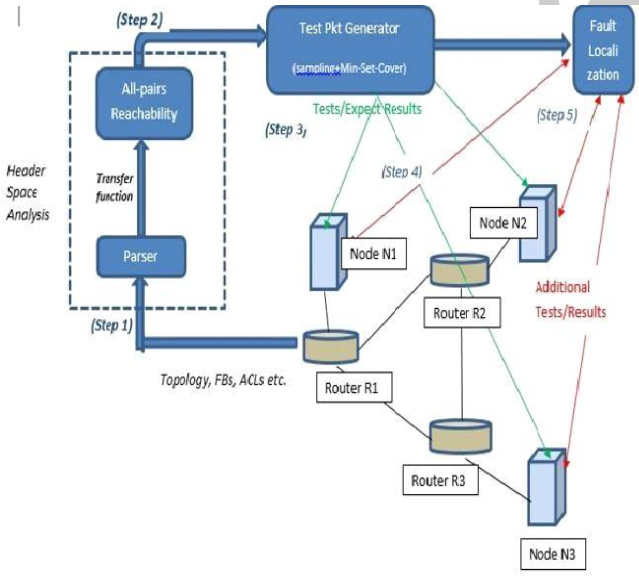



**Fig 2: ATPG block diagram**

### *2. GENERATING TABLE WITH ALL-PAIR REACHABLECONDITIONS*

While the packet is transferred in between one test terminal to another test station, the tool starts working by calculating the total bunch of packet headers in the first step. Then it computes an entire rule sets so that it exercises along the path from each such packet header. It takes use of all-pair reachability algorithm described in order to discover the possible pairs.

### I. Sampling

In the next step, the tool selects minimum of a test packet to act upon all the reachability rules. Selecting a lone packet per class in some random manner is the simplest mechanism.

### II. Compression:

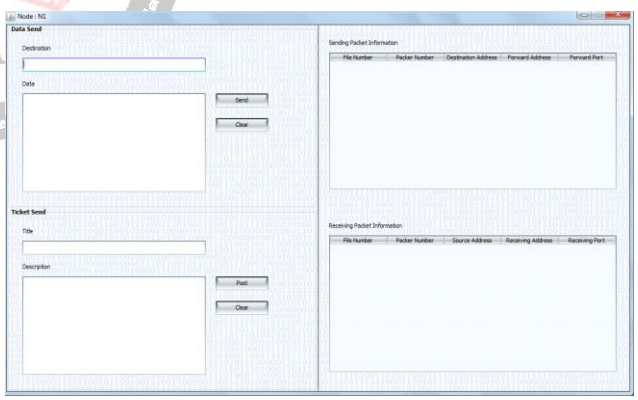
In this step, the packets are compressed to minimum number. So, the system selects a least set of packets from the sampling step, such that it covers all the rules of the merger of rule histories. The cover is selected in such a way that all connection or all router configurations are covered. This is known as Min-Set-Cover problem. While NP- Hard, greedy O(N2) gives a decent approximation, here N represents thetest packets. The obtained lowest set of test packets is referred as regular test packets fp1,p2, p3, p4, p5g and the remaining packets which are not selected comes under reserved test packetsfp6g.

## 2. LOCALIZATION OFFAULTS

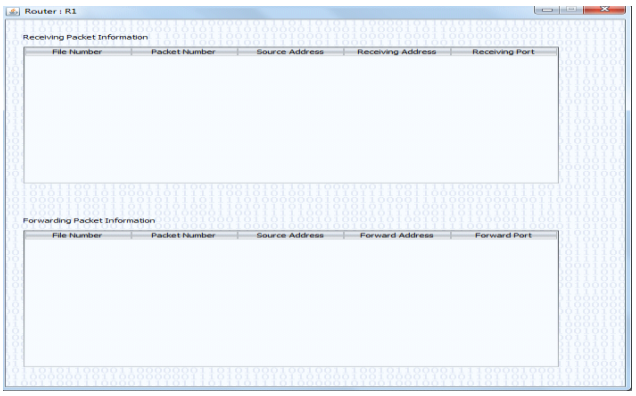
After sending the minimum set of test packets across the node, it tests for failures. If any faults are present in the path, the system diagnosis the defects. A bunch of test packets are sent sporadically by ATPG. If test packets fail, ATPG pinpoints the fault(s) that caused the problem. A rule fails if its experimental nature differs from its expected nature. ATPG keeps path of where rules fail using a result function "Success" and "failure" depend on the behaviour of the rule: A forwarding rule fails if a test packet is not delivered to the proposed output port, where a drop rule behaves correctly when packets are dropped. Similarly, a link failure is a interruption of a forwarding rule in the topology function. On the other hand, if an output link is congested, failure is obtained through the latency of a test packet going above a threshold

# VIII.DESIGN DETAILS

NODE 1.

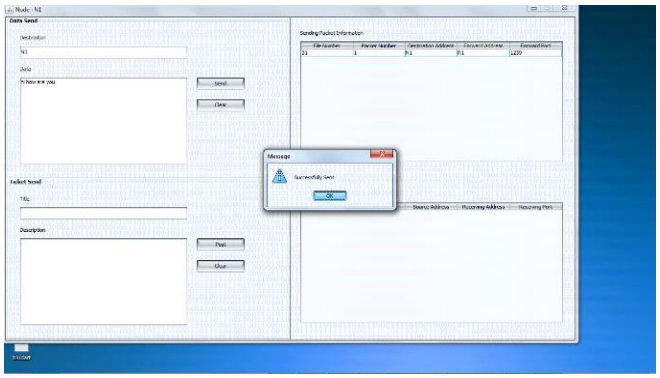



ROUTER 1



PACKET SEND



# IX. CONCLUSION

We ave tried to implement the paper "Automatic Test Packet Generation" IEEE Trans April 2014 and according to the implementation the conclusion is testing liveness of a network is a vital problem for huge data centre operators. Sending inquiries between every couple of edge ports is neither comprehensive nor accessible. It be sufficient to find a least set of end to end packets that navigate each link. However, undertaking this necessitates a way of extracting across device particular configuration files, generating headers and the links they touch, and finally determining a least set of test packets. Even the crucial problem of automatically generating test packets for efficient liveness testing wants techniques akin to ATPG.

ATPG, however, goes much auxiliary than liveness testing with the same framework. ATPG can test for approachable policy and performance health. In implementation also amplifies testing with a simple fault localization scheme also assembled using the header space framework. As in software testing, the formal model helps exploit test coverage while diminishing test packets. the outcomes declares that all forwarding rules in Stanford backbone or Internet2 can be applied by a amazingly lesser number of test packets.

Network managers today use primitive tools such as and. in the survey results indicate that they are eager for more sophisticated tools. Other fields of engineering indicate that these desires are not unreasonable: For example, both the ASIC and software design industries are buttressed by billion-dollar tool businesses that supply techniques for both static and dynamic verification. ATPG was a well-known acronym in hardware chip testing, where it stands for Automatic Test *Pattern* Generation . the network ATPG will be equally useful for automated dynamic testing of production networks.

# REFERENCES

[1] N. Duffield, F. L. Presti, V. Paxson, and D. Towsley, "Inferring link loss using striped unicast probes," in *Proc. IEEE INFOCOM*, 2001, vol. 2, pp. 915–923.

[2] N. Duffield, "Network tomography of binary network performance characteristics," *IEEE Trans. Inf. Theory*, vol. 52, no. 12, pp. 5373–5388, Dec. 2006.

[3] Y. Bejerano and R. Rastogi, "Robust monitoring of link delays and faults in IP networks," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 1092–1103, Oct. 2006.

[4] C. Cadar, D. Dunbar, and D. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *Proc. OSDI*, Berkeley, CA, USA, 2008, pp. 209–224.

[5] M. Canini,D.Venzano, P. Peresini,D.Kostic, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. NSDI*, 2012, pp. 10: 2010.

[6] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data," in *Proc. ACM CoNEXT*, 2007, pp. 18:1/18/12.

[7] "Automatic Test Pattern Generation," 2013 [Online].Available: http://en.wikipedia.org/wiki/Automatic test pattern generation.

[8] P. Barford, N. Duffield, A. Ron, and J. Sommers, "Network performance anomaly detection and localization," in *Proc. IEEE INFOCOM*, April pp. 1377–1385.

[9] "Beacon," [Online]. Available: http://www.beaconcontroller.net/

[10] "ATPG code repository," [Online]. Available: http://eastzone.github.com/atpg/